INRIA

UNITÉ DE RECHERCHE
INRIA-LORRAINE

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

1992

25ème anniversaire

N° 1795

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

DÉVELOPPEMENT DE L'ALGORITHME D'UNIFICATION DANS LE CALCUL DES CONSTRUCTIONS AVEC TYPES INDUCTIFS

Joseph ROUYER

Novembre 1992

RR-1795

Développement de l'algorithme d'unification dans le Calcul des Constructions avec types inductifs.

Development of the unification algorithm in the Calculus of Constructions with inductive types

Joseph Rouyer

Centre de Recherche en Informatique de Nancy (CNRS)
and INRIA-Lorraine
Campus Scientifique, BP 239,
54506 Vandœuvre-lès-Nancy, France

email: Joseph.Rouyer@loria.fr

Résumé

On trouve dans cet article une traduction possible de la notion de termes et de substitution dans le Calcul des Constructions avec Types Inductifs et une preuve constructive de l'unification au premier ordre de laquelle on peut extraire un algorithme (proche de celui de Robinson). Le logiciel utilisé pour le codage et la vérification des preuves est le Coq Proof Assistant version 5.6. L'article est conçu pour une lecture en parallèle du code en Coq et de la transcription qui en est faite en langage mathématique. Il a été écrit également dans le but d'aider ceux qui le désirent à pénétrer rapidement dans le monde du Calcul des Constructions.

Abstract

One finds in this paper a possible translation of the notion of terms and substitutions in the Calculus of Constructions with Inductive Types and a constructive proof of the first order unification form. This proof yields an algorithm which turns out to be close to this of Robinson. The software used for the coding and the verification of the proofs is the Coq Proof Assistant version 5.6. The paper is designed for a parallel reading of the Coq code and of its transcription in mathematical language. It can also be used used as an help for those who wish to enter quickly the world of the Calculus of Constructions. (This report is in French.)

Développement de l'algorithme d'unification dans le Calcul des Constructions avec types inductifs.

Joseph Rouyer

17 Septembre 1992

Abstract

On trouve dans l'article qui suit une traduction possible de la notion de termes et de substitution dans le Calcul des Constructions avec Types Inductifs, une preuve constructive de l'unification au premier ordre de laquelle on peut extraire un algorithme (proche de celui de Robinson). Le logiciel utilisé pour le codage et la vérification des preuves est le Coq Proof Assistant version 5.6. L'article est conçu pour une lecture en parallèle du code et de la transcription qui en est faite en langage mathématique. Il a été écrit également dans le but d'aider ceux qui le désirent à pénétrer rapidement dans le monde du Calcul des Constructions.

1 Introduction

Le premier objectif de cet article est de présenter une preuve de l'unification au premier ordre dans un ensemble de termes, réalisée dans le Calcul des Constructions avec types inductifs, en utilisant l'implantation de celui-ci faite dans le Coq Proof Assistant version 5.6. Dans tout ce qui suit "C.C." est une abréviation de Calcul des Constructions. Le lecteur est supposé avoir à sa disposition le manuel de référence [1].

Le C.C. avec types inductifs fournit un cadre formel dans lequel on peut faire des démonstrations sans avoir nécessairement recours à une machine. A fortiori toute démonstration utilisant le Coq Proof Assistant version 5.6 peut donc être reprise et formulée comme n'importe quelle preuve mathématique. Les démonstrations présentées ici ne sont pas toujours les plus rapides mais elles ont été contrôlées. Un de nos problèmes est de les rendre claires. Le point fort du Coq Proof Assistant est qu'il permet de construire inductivement des objets puis de mettre en place n'importe quel raisonnement par induction et même d'en imbriquer plusieurs. En cela, il peut être une aide réelle au raisonnement. On peut aussi s'en servir pour

mieux poser donc mieux appréhender certains problèmes. De plus, la possibilité de faire appliquer des résultats de manière automatique (commande "*Auto*") peut faire apparaître des possibilités auxquelles l'utilisateur n'avait pas pensé auparavant.

On peut donc voir dans tout ce qui suit une preuve de l'unification au premier ordre dans un ensemble de termes dans la logique du C.C. avec types inductifs ou bien le commentaire de la réalisation de cette preuve dans le Coq Proof Assistant version 5.6. avec des remarques techniques sur l'utilisation de ce logiciel.

Le deuxième objectif de l'étude est d'extraire de la preuve un programme en FML ou CAML. L'extraction en FML se fait selon la méthode basée sur la notion de réalisabilité définie par Christine Paulin-Mohring dans le C.C. [3]. Cette méthode fut implantée une première fois dans la version 4.10 du C.C.. Le mécanisme d'extraction prévu dans le Coq Proof Assistant version 5.6 est toujours basé sur la même notion de réalisabilité.

Le type *Prop* est, dans le système, le type des propositions purement logiques (dont on ne veut rien extraire), tandis que *Set* est le type des ensembles, des spécifications, ainsi que, plus généralement, celui des propositions dont le contenu constructif se retrouvera, au moins en partie, dans le programme extrait. C'est pourquoi on trouvera des théorèmes ou des lemmes de type *Set* aussi bien que de type *Prop*.

1.1 Problème posé par la notion de terme

A cause de restrictions dues au C.C., on utilisera dans la suite une notion inhabituelle et plus générale que la notion de termes, que nous apellerons les "quasi-termes". On démontrera que, ou bien deux quasitermes sont unifiables et il existe un unificateur plus général au sens habituel, ou bien les deux quasitermes ne sont pas unifiables (c'est à dire qu'aucune substitution ne les rend égaux). Les termes seront définis comme des quasitermes particuliers et on démontrera que s'ils sont unifiables comme quasitermes, leur unificateur principal est une substitution qui produit des termes quand elle s'applique à des termes.

Un ensemble de termes est habituellement défini de manière inductive. La définition fait alors intervenir un ensemble fini de symboles, noté ici Σ, une application, notée ici *arité*, de Σ dans l'ensemble *nat* des entiers naturels et un ensemble au plus dénombrable de symboles indexés ou non ne figurant pas dans Σ, noté ici *var*.

Définition 1 *Les termes sur Σ avec var comme ensemble de variables sont définis inductivement par les assertions suivantes :*

1. *un élément v de var est un terme (on dit que v est une variable),*
2. *si c est un élément de Σ tel que arité(c) = 0, alors c est un terme (on dit que c est une constante).*

3. *si f est un élément de Σ tel que arité(f) > 0 et si l est une liste de termes de longueur arité(f), alors $f(l)$ est un terme.*

Définition 2 *(variante) Les termes sur Σ avec var comme ensemble de variables sont définis inductivement par les assertions suivantes :*

1. *un élément v de var est un terme (on dit que v est une variable),*

2. *si f est un élément de Σ telle que arité(f) $= n$ et si l est une liste de termes de longueur n, alors $f(l)$ est un terme.*

La traduction dans le C.C. avec types inductifs de chacune de ces définitions par une définition d'un type inductif de type ensemble (*Set*) est impossible pour deux raisons principales.

Tout d'abord, on ne dispose pas de la notion de sous-type pour définir les variables et les constantes.

Ensuite dans les assertions *3)* de la première définition et *2)* de la deuxième, *termes* est dans une **position dite non positive**, (positive étant employé ici avec le sens que lui donne Christine Paulin dans sa thèse [3]). Or dans le C.C. avec types inductifs ou non, il n'est pas permis pour un objet A de le définir inductivement à l'aide d'un constructeur dans le profil duquel A n'apparait pas positivement, ce qui serait le cas avec

$$Enracinement \ : \ \Sigma \longrightarrow liste(terme) \longrightarrow terme,$$

en utilisant la définition polymorphique de *liste* du C.C..

Enfin le **contrôle du respect des arités** pose un problème insoluble si on tient à rester dans le type *Set* (il y a toutefois une possibilité passant par l'utilisation des types dépendants, mais qui exclut l'extraction). A cause de tout cela, on définira une structure de donnée intermédiaire qui sera l'ensemble des quasitermes. Dans les deux définitions précédentes on supprime tout ce qui est contrôle pour ne conserver que les notions constructives. De plus on remplace la notion de listes de termes par celle de listes non vides de quasitermes qui sont aussi des quasitermes. Plus précisément, les constructeurs correspondent à l'injection des variables dans les quasitermes, à celle des constantes, à l'enracinement des symboles fonctionnels et à une fabrication de couples qui correspond à l'ajout en tête dans une liste et que l'on peut appeler combinaison.

Définition 3 *Les quasitermes sur Σ avec var comme ensemble de variables sont définis inductivement par les assertions suivantes :*

1. *un élément v de var est un quasiterme (on dit que v est une variable).*

2. *si c est un élément de Σ, alors c est un quasiterme. (on dit que v est une constante),*

3. *si f est un élément de* Σ *et si t est un quasiterme, alors f(t) est un quasiterme.*

4. *si t et t' sont deux quasitermes, alors (t,t') est un quasiterme.*

Pour justifier l'approche des termes par les quasitermes, on peut montrer que les termes tels qu'on les conçoit habituellement forment un sous ensemble de l'ensemble des quasitermes. On procède pour cela en deux temps en appelant *terme* et $l-terme$ ce que Manna et Waldinger nomment *expressions* et $l-expressions$ dans [2].
On se donne une fonction *arité* de Σ dans l'ensemble des entiers naturels.
On définit une fonction *longueur* de l'ensemble des quasitermes vers celui des entiers naturels de la manière suivante: si t est une variable ou une constante ou un quasiterme enraciné c'est à dire de la forme $f(u)$, alors la longueur de t est 1, sinon t est de la forme (u, u') et sa longueur est la somme de celle de u et de celle de u'.
On dit qu'un quasiterme t est *simple* si t n'est pas de la forme (t, t') autrement dit si t est de longueur 1.
On définit ensuite la notion de liste-terme puis celle de terme.

Définition 4 *On dit qu'un quasiterme t est un liste-terme dès qu'il vérifie une des assertions suivantes :*

1. *t est une variable,*

2. *t est une constante,*

3. *t=f(l) où l est un liste-terme dont la longueur est égale à l'arité de f,*

4. *t=(u,u') où u est un liste-terme simple et u' un liste-terme quelconque.*

Définition 5 *On dit qu'un quasiterme t est un terme si t est un liste-terme simple.*

1.2 Le problème de l'unification

On définit habituellement une substitution s dans l'ensemble des termes comme un couple (D,σ) où D est un sous-ensemble fini de *var* et σ une application de D dans l'ensemble des termes.

Définition 6 *Etant donnée une substitution* (D,σ) *l'image d'un terme par s est définie inductivement par*

1. *si x est dans D, alors* $s(x)=\sigma(x)$,

2. *si x n'est pas dans D, alors* $s(x)=x$,

3. *si c est un élément de* Σ, $s(c)=c$,

4. *si nil est la liste vide,* $s(nil)=nil$.

5. *si* $l=cons(t_0, l_0)$. $s(l)=cons(s(t_0).s(l_0))$,

6. *si f est un élément de* Σ *et si l est une liste de termes,* $s(f(l))=f(s(l))$.

La définition précédente fait intervenir des contrôles. A cause de cela on lui préférera une autre démarche pour définir une substitution dans les quasitermes.

Définition 7 *Si* σ *est une application de var dans l'ensemble des quasitermes, on appelle substitution associée à* σ *l'application s de l'ensemble des quasitermes dans lui-même inductivement définie par*

1. $s(v)=\sigma(v)$.

2. $s(c)=c$.

3. $s(f(l))=f(s(l))$

4. $s((t,t'))=(s(t).s(t'))$.

Le problème de l'unification se posera alors dans l'ensemble des quasitermes dans les termes suivants.

Définition 8 *On dit que les quasitermes t et t' sont unifiables s'il existe une substitution s telle que* $s(t)=s(t')$.

Définition 9 *Résoudre le problème de l'unification des quasitermes t et t' c'est ou bien trouver une subsitution s telle que* $s(t)=s(t')$
ou bien montrer que pour toute subsitution s, $s(t)\neq s(t')$.

Avant de terminer ce paragraphe, on peut faire deux remarques qui peuvent être négligées en première lecture.
La première est que la notion de quasitermes est assez loin de celle de termes et vient de choix arbitraires. On aurait pu tout aussi bien poser comme definition :

Définition 10 (non retenue)
Les quasitermes sur Σ *avec var comme ensemble de variables sont définis inductivement par les assertions suivantes :*

1. *un élément v de var est un quasiterme (on dit que v est une variable)*

2. *si c est un élément de* Σ, *alors c est un quasiterme, (on dit que c est une constante)*

3. *si f est un élément de* Σ *et si t est un quasiterme, alors f(t) est un quasiterme,*

4. *si t est un quasiterme et v un élément de var, alors (v,t) est un quasiterme,*

5. *si t est un quasiterme et c un élément de* Σ *alors (c,t) est un quasiterme,*

6. *si f est un élément de* Σ, *si t et t' sont deux quasitermes alors (f(t).t') est un quasiterme.*

Cette définition a l'avantage d'alléger le contrôle de bonne formation des listes d'arguments. Elle interdit la formation d'objets comme $(((v,c),(f,f(v))),(v,c))$ où v est une variable, c et f deux éléments de Σ. Elle a deux inconvénients. D'abord, les assertions 4), 5) et 6) renferment deux actions habituellement bien séparées, qui sont l'ajout en tête d'un objet dans une liste et un contrôle sur la forme de cet objet. Enfin, la traduction en C.C. de cette notion de quasitermes par une définition inductive va comporter deux constructeurs de plus, dont le dernier (celui qui correspond à l'assertion 6)) est assez complexe, ce qui, d'un point de vue pratique, se traduira par la multiplication de la longueur des preuves par un nombre voisin de deux, par le carré de ce nombre, par son cube etc... , selon le nombre de quasitermes sur lesquels on fait un raisonnement par induction. On peut rappeler ici que L.C. Paulson a choisi pour définir sa notion de *term* dans [4] une structure encore plus simple que celle des quasitermes retenue plus haut, puisqu'il n'utilise que trois constructeurs qui sont $CONST$, VAR et $COMB$ où

- $CONST$: $\Sigma \longrightarrow term$,
- VAR : $var \longrightarrow term$ et
- $COMB$: $term \longrightarrow term \longrightarrow term$.

Cette façon de voir regroupe en une même action l'ajout en tête dans une liste et l'enracinement, option qui n'a pas été choisie ici mais qui le sera ultérieurement. Elle demande en effet assez peu de transformations et le programme extrait sera sans doute plus court. La raison principale de ce choix est qu'avec la structure de données utilisée par L.C. Paulson la notion de sous-terme ne se conserve pas. Par exemple $COMB(f,t)$ admet f et t comme sous-termes, alors que le terme $f(t)$ n'admet que t comme sous-terme, ce qui dans d'autres problèmes que l'unification peut être un handicap.

La seconde remarque est que la notion de quasiterme est suffisamment loin de celle de terme pour que la résolution du problème de l'unification dans les quasitermes n'implique pas sa résolution dans l'ensemble des termes. Il faut s'assurer qu'étant donné deux termes t et t' l'existence d'une substitution s telle que $s(t) = s(t')$, implique que $s(t)$ donc $s(t')$ sont également des termes et que s associe un terme à toute variable de son domaine.

1.3 Plan suivi

Le plan suivi est celui du programme, avec plusieurs petits paragraphes sur le C.C. ou sur son implantation, intercalés et nettement détachés. Ce sont des remarques sur les méthodes suivies et le fonctionnement du Coq Proof Assistant version 5.6. Toute définition ou tout théorème énoncé correspond à une définition ou à un théorème dans le script auquel le lecteur est invité à se reporter. Une lecture parallèle du script et du texte est fortement recommandée.

La vérification du fait que la résolution du problème de l'unification dans les quasitermes implique sa résolution dans l'ensemble des termes n'apparait pas dans le texte qui suit mais elle apparait tout au long du script encadrée par
"(*———— Verifications on the terms begin ————*)" et par
"(*———— Verifications on the terms end ————*)".

La section 8 contiendra un résumé et des remarques sur ces vérifications.

La section 2 est consacrée à la définition des quasitermes puis à des considérations plus générales sur l'égalité dans le C.C. ainsi que sur les définitions inductives et récursives.

Dans la section 3, on étudie les variables des quasitermes. Après avoir traduit la notion simple "être dans un terme" pour une variable, on revient sur le fonctionnement du système pour montrer au moins sur un exemple comment fonctionne la commande "Elim" qui correspond à l'élimination en C.C.. On essaie ensuite de traduire dans le C.C. la notion de substitution.

La section 4 est consacrée aux listes de variables et à l'introduction de prédicats correspondant à des notions simples mais qui sont plutôt des notions ensemblistes : nombre des différents éléments, suppression des occurences d'un élément donné, inclusion large et inclusion stricte. Vient ensuite une comparaison des définitions inductives avec les définitions récursives. La section se termine avec la définition de la liste des variables d'un quasiterme et de prédicats liés aux variables sur les quasitermes et les substitutions.

Dans la section 5 on étudie la relation sous-quasiterme strict et ses liens avec les substitutions.

La section 6 est consacrée à l'étude de l'idempotence dans l'ensemble des substitutions et à la composition des substitutions. C'est dans cette section qu'est mis en évidence le fait que les substitutions idempotentes diminuent dans certaines conditions le nombre des variables.

Dans la section 7, on traduit la notion d'unificateur minimal idempotent et on spécifie le problème de l'unification. Les cas d'échec par le symbole de tête sont étudiés rapidement ainsi que le cas où un des quasitermes est une variable, celui où c'est une constante, celui où les deux quasitermes ont le même constructeur de tête. Avant le théorème final, on énonce les derniers résultats sur les quasitermes clos, le nombre des variables distinctes, la composition des unificateurs minimaux, y compris dans le cas où l'un d'eux est la subsitution de domaine vide.

2 Définition des quasitermes dans le C.C. avec types inductifs

Σ est remplacé par un ensemble *fun* au plus dénombrable.

Définition 11 *Soient var et fun deux ensembles au plus dénombrables. Les quasitermes sur var et fun sont définis inductivement par les assertions suivantes :*

1. *à tout élément x de var correspond un quasiterme $V(x)$,*
2. *à tout élément f de fun correspond un quasiterme $C(f)$,*
3. *si f est un élément de fun et si t est un quasiterme, alors $Root(f,t)$ est un quasiterme.*

4. si t_1 et t_2 sont deux quasitermes, alors $ConsArg(t_1,t_2)$ est un quasiterme.

On peut voir dans V et dans C une manière d'indexer ($V(n)$ pour V_n et $C(n)$ pour C_n). On choisira donc *nat* comme ensembles *var* et *fun*, qui n'ont pas à être distincts, puisque V et C permettent de différencier les quasitermes variables des quasitermes constantes.
On notera QT l'ensemble des quasitermes. t étant un élément de QT, on a donc :

$$t ::= V(v) \mid C(f) \mid Root(f,t) \mid ConsArg(t,t).$$

2.1 Egalités, définitions inductives et récursives

L'égalité utilisée dans les objets de type ensemble (Set) du C.C. avec types inductifs est définie par un type dépendant d'un paramètre à valeur dans Set. Plus précisément, pour un A de type Set et un a dans A donnés, c'est l'égalité à a qui est définie (relation unaire sur A). Elle l'est inductivement à l'aide d'un seul constructeur $refl\_equal(A,a)$ de type $a = a$. La réflexivité de l'égalité dans A (relation binaire sur A) est donc immédiate.
De plus, le schéma de récursion de l'égalité à a appliqué à un prédicat P sur A et à un élément b de A se réduit en $P(a) \Longrightarrow P(b)$. Donc, dans la pratique si on élimine une hypothèse h de type $a = b$ sur un but $P(b)$, on obtient $P(a)$ comme nouveau but.

L'égalité dans A est donc l'égalité au sens de Leibniz : a et b sont égaux dans l'ensemble A signifie que pour tout prédicat P sur A, $P(a) \Longrightarrow P(b)$. On peut aussi remarquer que si R est une relation réflexive dans A, $a = b \Longrightarrow R(a,b)$, puisque l'élimination de l'hypothèse de type $a = b$ remplace le but $R(a,b)$ par le but $R(a,a)$. Cela prouve que l'égalité dans A est la plus petite relation binaire réflexive dans A.

Le Coq Proof Assistant version 5.6, dans laquelle on a développé la preuve, contient dans son environnement un certain nombre de résultats concernant cette égalité (il s'agit de la réflexivité, de la symétrie et de la transitivité) ainsi que des outils, appelés tactiques dans le système, permettant de les utiliser facilement (*Reflexivity, Symmetry, Transitivity et Replace*). Enfin, l'environnement permet l'application automatique de la réflexivité et, dans le cas où cela permet de conclure en un seul pas, de la symétrie.

Si un type A de type Set a été défini (on peut dire aussi construit) **inductivement** il suffit d'utiliser la commande *Elim t* pour t dans A pour faire apparaître les buts correspondant au raisonnement par induction sur la structure de t. Par exemple, pour $P : QT \rightarrow Prop$, si on élimine t sur $P(t)$ il apparait quatre sous-buts :

- sous-but1 : quel que soit v dans var, $P(V(v))$,
- sous-but2 : quel que soit f dans fun, $P(C(f))$,

- sous-but3 : quels que soient f dans fun et t dans QT, $P(t)$ implique $P(Root(f,t))$.
- sous-but4 : quels que soient t et t' dans QT, si $P(t)$ et si $P(t')$, alors $P(ConsArg(t,t'))$.

Enfin si un type T de type Set a été **construit inductivement**, on dispose de l'opération $match$ qui permet de définir des applications de T vers $Prop$ ou vers n'importe quel type de type $Prop$ ou de type Set en associant à chaque élément t de T des termes du C.C. par une définition récursive sur t.
Supposons que $K = Prop$ ou $K : Prop$ ou $K : Set$, et, que a, b, g et h sont des applications dont les profils respectifs sont :
$a : var \to K$, $b : fun \to K$, $g : fun \to QT \to K \to K$,
et $h : QT \to K \to QT \to K \to K$.
Si on veut définir une application $F : QT \to K$ telle que

- $F(V(x))$ se simplifie en $a(x)$ pour tout variable x,
- $F(C(f))$ en $b(f)$ pour tout f dans fun,
- $F(Root(f,t))$ en $g(f,t,F(t))$ pour tout f dans fun et tout quasiterme t, et,
- $F(ConsArg(t,t'))$ en $h(t,F(t),t',F(t'))$, pour tout quasiterme t et tout quasiterme t',

on doit définir F comme le filtrage sur t ($(< K > Match\ t\ with \ldots)$ dans le système) avec

- a,
- b,
- $\lambda f{:}fun.\lambda t{:}QT.\lambda k{:}K.g(f,t,k)$ et
- $\lambda t{:}QT.\lambda k{:}K.\lambda t'{:}QT.\lambda k'{:}K.h(t,k,t',k')$.

Cette possibilité sera utilisée dès les premières définitions pour définir des prédicats structuraux et des destructeurs. Dans la suite l'opération "match" sera appelée filtrage et toute définition la faisant intervenir sera appelée définition récursive. Le mot "récursive" ne sera pas employé en dehors de ce cas.

Exemple: Si on veut définir une fonction qui, appliquée à $ConsArg(t_1,t_2)$, donne t_1, on la définit comme effectuant un filtrage sur t avec

- V ou n'importe quelle fonction de var dans QT,
- C ou n'importe quelle fonction de fun dans QT,
- $\lambda f{:}fun.\lambda t{:}QT.\lambda u{:}QT.t$ ou toute autre fonction de profil $fun \to QT \to QT \to QT$
- $\lambda t{:}QT.\lambda u{:}QT.\lambda t'{:}QT.\lambda u'{:}QT.t$.

2.2 Traitement de l'égalité dans le type des quasitermes

Pour pouvoir manipuler les égalités dans l'ensemble QT des quasitermes, on a respecté l'ordre suivant :

1. définition inductive du type QT de type Set,
2. construction des prédicats portant sur la nature du constructeur de tête,
3. construction récursive des destructeurs associés aux constructeurs de QT,
4. lemmes de destruction démontrés à l'aide de ces destructeurs et permettant par la suite de ne plus les utiliser,
5. énoncés et démonstrations des lemmes de non-égalité,
6. décidabilité (aux niveaux Set et $Prop$) de l'égalité dans QT.

Par lemmes de destruction, on entend des résultats de la forme :
"si $C(a_1,\ldots,a_i,\ldots,a_n) = C(b_1,\ldots,b_i,\ldots,b_n)$, alors $a_i = b_i$ pour $1 \leq i \leq n$".
Par lemmes de non-égalité, on entend des résultats de la forme :
"s'il existe i tel que $a_i \neq b_i$", alors $C(a_1,\ldots,a_i,\ldots,a_n) \neq C(b_1,\ldots,b_i,\ldots,b_n)$.

Les lemmes de destruction permettent dans bien des cas d'appliquer les hypothèses de récurrence dans les raisonnements par induction.
Les lemmes de non-égalité sont utiles dans les raisonnements par l'absurde, dans les preuves de décidabilité et lors de l'utilisation des définitions où figurent des négations.
Enfin, les résultats sur la décidabilité de l'égalité permettent les raisonnements par cas.

Les lemmes de destruction sont les suivants :

Lemme 1
Quels que soient les éléments x_1 et x_2 de var, si $V(x_1)=V(x_2)$, alors $x_1=x_2$.
Quels que soient les éléments f_1 et f_2 de fun, si $C(f_1)=C(f_2)$, alors $f_1=f_2$.
Quels que soient les symboles f_1 et f_2 et les quasitermes t_1 et t_2, si $Root(f_1,t_1)=Root(f_2,t_2)$, alors $f_1=f_2$.
Quels que soient les symboles f_1 et f_2 et les quasitermes t_1 et t_2, si $Root(f_1,t_1)=Root(f_2,t_2)$, alors $t_1=t_2$.
Quels que soient les quasitermes t_1, t_2, u_1 et u_2, si $ConsArg(t_1,u_1)=ConsArg(t_2,u_2)$, alors $t_1=t_2$.
Quels que soient les quasitermes t_1, t_2, u_1 et u_2, si $ConsArg(t_1,u_1)=ConsArg(t_2,u_2)$, alors $u_1=u_2$.

Des lemmes de destruction, on déduit les lemmes de non-égalité, qui eux-mêmes seront utilisés pour démontrer la décidabilité de l'égalité dans QT.

Théorème 1 *Quels que soient les quasitermes t_1 et t_2, la proposition "t_1=t_2" est décidable.*

La démonstration de ce théorème est stucturée par une induction sur t_1 et une induction sur t_2. Elle utilise en plus la décidabilité de l'égalité dans *var* et dans *fun*.

3 Variables dans un quasiterme et substitutions

Les définitions et les résultats qui suivent, même s'ils sont évidents ou implicites, sont rappelés sous la forme qui permet leur traduction facile et fidèle en C.C. avec types inductifs.

Définition 12 *(définition inductive)*
Soient x une variable et t un quasiterme. La relation "x est dans" définie sur QT est la plus petite relation sur QT vérifiant

1. *x est dans V(x),*
2. *si x est dans t', alors x est dans Root(f,t'),*
3. *si x est dans t_1, alors x est dans ConsArg(t_1,t_2),*
4. *si x est dans t_2, alors x est dans ConsArg(t_1,t_2).*

Définition 13 *(variante récursive)*
Soit x une variable. Le prédicat qui à tout quasiterme t associe "x est dans t" est le prédicat défini par filtrage sur t qui se simplifie en

1. *$x = x_0$, si $t = V(x_0)$,*
2. *False, si $t = C(f)$.*
3. *"x est dans t'", si $t = Root(f,t')$,*
4. *"x est dans t_1 ou x est dans t_2", si $t = ConsArg(t_1,t_2)$.*

Lemme 2

1. *Les deux définitions précédentes sont équivalentes.*
   ("x est dans t" sera noté is-in(x,t) lorsqu'on utisera la définition inductive, et IS-IN(x,t) lorsqu'on utilisera la définition récursive).
2. *Quels que soient la variable x et le quasiterme t, la proposition "x est dans t" est décidable.*

Les démontrations se font par induction sur t et utilisent la décidabilité de l'égalité dans *var*.

3.1 Elimination des hypothèses dont le type est inductif

Soit un prédicat P sur un type T défini inductivement. Si on utilise "*Elim h*" sur le but $BUT(t)$ avec $h : P(t)$, le schéma de récursion de P est appliqué à $BUT(t)$. Que se passe-t-il alors ?
P est le plus petit prédicat sur T vérifiant un certain nombre d'assertions $A_1, \ldots, A_n$. Cela signifie que pour tout autre prédicat Q sur T, si $A_1[Q/P], \ldots, A_n[Q/P]$ et si $P(t)$ sont vrais, alors $Q(t)$ est vrai. Mais si $P(t)$ est vrai, $Q(t)$ est vrai si et seulement si $P(t) \wedge Q(t)$ est vrai. Donc si $P(t)$ est vrai, pour que $Q(t)$ soit vrai il suffit que $A_1[(\lambda t.P(t) \wedge Q(t))/P], \ldots, A_n[(\lambda t.P(t) \wedge Q(t))/P]$ le soient.
Par ailleurs, compte tenu de la syntaxe des contructeurs de types inductifs, chaque A_i est soit de la forme $P(c_0)$ où c_0 est un constructeur de T, soit de la forme $X_1 \rightarrow \cdots \rightarrow X_p \rightarrow P(t_i)$ où t_i est une expression de type T.
Lors de l'élimination d'une hypothèse $h : P(t)$, l'opérateur de récursion de P noté *P_ind* est appliqué à t, et à BUT. Le but $BUT(t)$ est alors remplacé par n sous-buts B_i pour i allant de 1 à n.

B_i est de la forme $BUT(c_0)$ si A_i était de la forme $P(c_0)$. Comme $P(c_0)$ est vrai, $BUT(c_0)$ équivaut alors à $P(c_0) \wedge BUT(c_0)$.

Sinon B_i est obtenu en remplaçant dans $A_i[(\lambda t.P(t) \wedge BUT(t))/P]$ le terme $P(t_i) \wedge BUT(t_i)$ qui termine la chaîne par $BUT(t_i)$, $P(t_i)$ étant nécessairement prouvé. En effet, les prémisses des B_i sont les mêmes que ceux des $A_i[(\lambda t.P(t) \wedge Q(t))/P]$, ils contiennent donc de quoi reconstituer, par élimination des "$\wedge$", les prémisses des A_i , c'est à dire les hypothèses qui prouvent $P(t_i)$.

Ceci est clair dans l'exemple qui suit. Soit $P = \lambda t.$*is-in(x,t)* , si on élimine *h:is-in(x,t)* sur $BUT(t)$, on obtient 4 sous-buts (autant que de constructeurs de $\lambda t.$ *is-in(x,t)*) :

- sous-but1 : $BUT(V(x))$.
- sous-but2 : quels que soient f dans fun et le quasiterme t_0, si *is-in(x,t0)* et si $BUT(t_0)$ alors $BUT(Root(f, t_0))$.
- sous-but3 : quels que soient les quasitermes t_1 et t_2, si *is-in(x,t1)*, et $BUT(t_1)$, alors $BUT(ConsArg(t_1, t_2))$.
- sous-but4 : quels que soient les quasitermes t_1 et t_2, si *is-in(x,t2)* et $BUT(t_2)$, alors $BUT(ConsArg(t_1, t_2))$.

Cela revient à prouver que le prédicat sur l'ensemble des quasitermes $\lambda t.BUT(t)$ est plus grand que $\lambda t.$ *is-in(x,t)* en montrant qu'il vérifie les quatre assertions que $\lambda t.$ *is-in(x,t)* est le plus petit prédicat à vérifier. Autrement dit, si on prouve les quatre sous-buts, on a prouvé que *is-in(x,t0)* implique $BUT(t)$.

3.2 Substitutions

Définition 14 *On appelle quasisubstitutions les applications de var vers QT.*

Ensuite, on définit récursivement sur la structure de t l'image d'un quasiterme t par la substitution associée à une quasisubstitution s.

Définition 15 *Soit s une quasisubstitution. On appelle substitution associée à s l'application Subst(s) de QT dans QT qui à tout quasiterme t associe le quasiterme Subst(s,t) défini par filtrage sur t qui se simplifie en*

1. *s(x) si t = V(x),*
2. *C(f) si t = C(f),*
3. *Root(f,Subst(s,t_0)) si t = Root(f,t_0),*
4. *ConsArg(Subst(s,t_1),Subst(s,t_2)) si t = ConsArg(t_1,t_2)).*

Dans le lemme qui suit, les démonstrations se font par induction sur t, par simplification de $Subst(s,t)$ et de $IS - IN(x,t)$, et par application des lemmes de destruction.

Lemme 3 *Soient t un quasiterme, s et s' deux substitutions.*

1. *Une variable x est dans Subst(s,t) si et seulement s'il existe y dans t tel que x soit dans s(y).*
2. *Les quasisubstitutions s et s' vérifient s(x) = s'(x), pour toute variable x de t. si et seulement si Subst(s,t) = Subst(s',t).*

4 Listes de variables

L'étude des listes est facile dans le C.C. avec types inductifs. Pour éviter un type dépendant, on définit directement les listes de variables en construisant le type inductif qui traduit cette notion sans utiliser de polymorphisme.

Définition 16 *Les listes d'éléments de var sont définies inductivement avec les deux constructeurs Nilv et Consv par :*

1. *Nilv est une liste d'éléments de var (Nilv est la liste vide).*
2. *si l est une liste d'éléments de var et si x est un élément de var, alors Consv(x,l) est une liste d'éléments de var.*

On définit récursivement par filtrage sur la liste l les prédicats $BConsv$ et $BNilv$ portant sur le symbole de tête de l, les fonctions $Headv$ donnant la tête et $Tailv$ donnant la queue de l, ainsi que la concaténation $Appv$ de deux listes l et l'.

Définition 17

1. *BNilv(l) se simplifie en True si l=Nilv et en False si l=Consv(x,l_0).*
   BConsv(l) se simplifie en False si l=Nilv et en True si l=Consv(x,l_0).
2. *Headv(x_0,l) se simplifie en x_0 si l=Nilv et en x si l=Consv(x,l_0).*
3. *Tailv(l) se simplifie en Nilv si l=Nilv et en l_0 si l=Consv(x,l_0).*
4. *Appv(l,l') se simplifie en l' si l=Nilv et en Consv(x,Appv(l_0,l')) si l=Consv(x,l_0).*

L'appartenance à une liste est définie de deux manières.

Définition 18 *(définition inductive)*
Soient listv l'ensemble des listes d'éléments de var et x un élément de var. "x est dans" est la plus petite relation définie sur listv qui vérifie

1. *x est dans Consv(x,l_0),*
2. *x est dans Consv(x_0,l_0) si x est dans l_0. On note is-in-lv(x,l) "x est dans l".*

Définition 19 *(version récursive)*
"x est dans l", noté IS-IN-LV(x,l), est le terme défini par filtrage sur l qui se simplifie

1. *en False si l=Nilv,*
2. *en ((x=x_0) ou IS-IN-LV(x,l_0)) si l=Consv(x_0,l_0).*

Lemme 4 *Soient x et x_0 deux éléments de var, l et l' deux listes.*

1. *Les propositions is-in-lv(x,l) et IS-IN-LV(x,l) sont équivalentes.*
2. *Si x≠ x_0 et si x est dans Consv(x_0,l) alors x est dans l.*
3. *Si x est dans Consv(x_0,nil), alors x=x_0.*
4. *"x est dans l" est décidable.*
5. *x est dans Appv(l,l') si et seulement si x est dans l ou x est dans l'.*

On démontre 1), 2) et 5) en raisonnant par induction sur *l*.
3) est une conséquence de la définition récursive.
4) utilise en plus la décidabilité de l'égalité dans *var*.

4.1 Compter les éléments distincts dans une liste

Pour compter une fois et une seule les différents éléments qui figurent dans une liste l. on définit un prédicat sur $liste \times nat$. Là encore, on le code de deux manières.

Définition 20 *(définition inductive)*
Le comptage des éléments différents d'une liste est la plus petite relation définie sur $listv \times nat$ *notée diffelnb qui vérifie*

1. *diffelnb(Nilv,O),*
2. *diffelnb(Consv(x,l),n) si diffelnb(l,n) et si x est dans l,*
3. *diffelnb(Consv(x,l),n+1) si diffelnb(l,n) et si x n'est pas dans l.*

On peut écrire une version récursive de cette définition.

Définition 21 *(définition récursive)*
Soient l une liste d'éléments de A et n un entier.
DIFFELNB(l,n) est le terme du C.C. avec types inductifs défini par filtrage sur l qui se simplifie

1. *en True si l=Nilv,*
2. *si* $l=Consv(x_0,l_0)$ *en un terme défini par filtrage sur n qui lui-même se simplifie en False si n=0 et en* $(((x_0$ *est dans* $l_0)$ *et* $DIFFELNB(l_0,q+1))$ *ou* $((x_0$ *n'est pas dans* $l_0)$ *et* $DIFFELNB(l_0,q)))$ *si* $n=q+1$.

Pour démontrer tout ce qui suit on utilise toujours des raisonnements par induction sur les listes et la décidabilité de "x est dans l".

Lemme 5 *Soient n un entier naturel, x une variable et l une liste de variables.*

1. *Les deux propositions diffelnb(l,n) et DIFFELNB(l,n) sont équivalentes.*
2. *Pour toute liste l, il existe un entier n tel que DIFFELNB(l,n).*
3. *Si DIFFELNB(l,0), alors l=Nilv.*
4. *Si DIFFELNB(Consv(x,l),n), alors n est non nul.*

4.2 Inclusion stricte au sens ensembliste

Dans les chapitres suivants on aura besoin d'une relation binaire dans $liste$ qui correspond à l'inclusion stricte dans les ensembles. On note $infl$ cette relation.

Définition 22 *(définition inductive)*
On dit que la liste l est strictement incluse dans la liste l' et on note infl(l,l') dès que toute variable dans l figure aussi dans l' et qu'il existe dans l' une variable qui n'est pas dans l.

Une définition récursive de cette relation n'a pas d'intérêt car complexe et trop difficile à simplifier par application du schéma de récursion. On pourrait dans un premier temps définir l'inclusion au sens large, ce qui est immédiat.

Lemme 6 *Aucune liste n'est strictement incluse dans Nilv.*

4.3 Suppression de toutes les occurences d'une variable dans une liste

Bien que lui aussi soit bien loin de la structure de liste, le prédicat qui suit peut avantageusement se définir des deux manières. Il s'agit d'exprimer que deux listes se déduisent l'une de l'autre par suppression des occurences d'une variable.

Définition 23 *(définition inductive)*
La relation ternaire notée remove est la plus petite relation définie sur var × listv × listv qui vérifie

1. *Pour tout x dans var, remove(x,Nilv,Nilv),*
2. *Pour tout x dans var, toute liste l et toute liste l' remove(x,l,l') entraine remove(x,Consv(x,l),l').*
3. *Pour tout x et tout x_0 dans var, pour tout l et tout l' dans listv , si $x \neq x_0$ et si remove(x,l,l'), alors remove(x,Consv(x_0,l),Consv(x_0,l')).*

remove(x, l, l') peut se lire "supprimer les occurences de x dans l donne l'". On retrouve là la manière "Prolog" des définition des fonctions.

Définition 24 *(définition récursive)*
Soient l et l' deux listes, x et x_0 deux éléments de var. REMOVE(x,l,l') est le terme du C.C. avec types inductifs défini par filtrage sur l qui se simplifie

1. *en (l'=Nilv) si l=Nilv.*
2. *si l=Consv(x_0,l_0), en (((x=x_0) et REMOVE(x,l_0,l')) ou ((x≠ x_0) et R')) où R' est le terme défini par filtrage avec l' qui se simplifie*
   2.1) en False si l'=Nilv,
   2.2) en ((x_0=x_1) et REMOVE(x,l_0,l_1)) si l'=Consv(x_1,l_1)).

On démontre en utilisant des raisonnements par induction sur les listes les résultats suivants.

Lemme 7

1. *Si x est une variable et si l et l' sont deux listes, les deux propositions remove(x,l,l') et REMOVE(x,l,l') sont équivalentes.*

2. *Si x n'est pas dans l et si REMOVE(x,l,l'), alors l=l'.*

3. *Si REMOVE(x,Consv(x,l),l'), alors REMOVE(x,l,l').*

4. *Si x et x_0 sont des variables distinctes, alors REMOVE(x,Consv(x_0,l),Consv(x_0,l_0)) entraine REMOVE(x,l,l_0).*

5. *Pour toute variable x et toute liste l, il existe une liste l_0 telle que REMOVE(x,l,l_0).*

6. *Si x et x_0 sont des variables distinctes, alors, quelles que soient les listes l et l_0, si REMOVE(x,Consv(x_0,l),l_0) alors l_0 n'est pas la liste vide et admet x_0 comme tête.*

4.4 Liens entre les IS-IN-LV, inflv, DIFFELNB et REMOVE

Le but du travail fait sur les listes est de pouvoir faire par la suite un raisonnement par récurrence portant sur le nombre des éléments distincts d'une liste de variables. On sera même amené à appliquer l'axiome de Peano généralisé: si un prédicat P sur *nat* est vrai pour 0 et est vrai pour n ($n \geq 1$) dès qu'il l'est pour tous les entiers strictement inférieurs à n, alors P est vrai pour tout entier.

Lemme 8

1. *Si REMOVE(x,l,l_0) et si x_0 est dans l_0, alors x_0 est également dans l.*

2. *Si x_0 est une variable de la liste l distincte de x et si REMOVE(x,l,l_0) alors x_0 est aussi une variable de la liste l_0.*

Les démonstrations de ces deux résultats se font en utilisant l'élimination des hypothèses de type $remove(x, l, l_0)$ version inductive de $REMOVE(x, l, l_0)$.

Les quatre théorèmes suivants sont longs et assez difficiles à démontrer, même s'ils énoncent des résultats facilement admis. Les deuxième et troisième théorèmes sont des conséquences du premier. Le dernier utilise le second théorème et l'axiome de Peano généralisé.

Théorème 2 *Etant données deux listes l et l_0, si l contient n éléments distincts (i.e. DIFFELNB(l,n)), si x est dans l et si REMOVE(x,l,l_0), alors l_0 contient n-1 éléments distincts.*

Théorème 3 *Etant donnés deux listes l et l_0 et deux entiers naturels n et n_0, si l contient n éléments distincts, si l_0 contient n_0 éléments distincts et si l est strictement incluse dans l_0, alors $n < n_0$.*

Plus généralement.

Théorème 4 *Etant donnés deux listes l et l_0 et deux entiers naturels n et n_0, si l contient n éléments distincts, si l_0 contient n_0 éléments distincts et si l est incluse dans l_0, alors $n_0 \leq n_1$.*

4.5 Définitions inductives et définitions récursives: une cohabitation fructueuse.

Plusieurs prédicats ont à la fois une définition inductive et une définition équivalente récursive. Chaque fois que cela est le cas, on note avec des lettres minuscules la définition inductive et avec des lettres majuscules la définition récursive. On peut faire à propos de ces deux formes de définitions deux remarques.
Tout d'abord, une définition récursive n'est intéressante que si la notion qu'elle veut traduire fait intervenir les constructeurs des types inductifs des objets auxquels elle s'applique, ce qui exclut en particulier la plupart des prédicats vrais s'il existe un objet d'un certain type vérifiant une certaine propriété.
D'autre part, pour éliminer une hypothèse h de type $DEFREC(t)$ où $DEFREC$ est une définition récursive, il faut d'abord éliminer t de façon à pouvoir simplifier $DEFREC(t)$, alors que l'élimination d'une hypothèse h de type $defind(t)$ où $defind$ est une définition inductive de la même notion est immédiate et peut donner lieu à moins de sous-buts.
Il faudra, par contre, éliminer t pour démontrer aussi bien le but $DEFREC(t)$ que le but $defind(t)$. Cependant, les simplifications de $DEFREC(t)$ que l'on peut alors effectuer vont permettre l'application des hypothèses de récurrence, alors que pour démontrer $defind(t)$, il est nécessaire de faire appel aux constructeurs de $defind$.
En conclusion, il est bon, pour la plupart des notions simples et d'utilisations fréquentes, de faire coexister les deux traductions de ces notions et de prouver leur équivalence.
Pour les démonstrations des deux théorèmes ci-dessus, on a remplacé l'hypothèse de type $DIFFELNB(l,n)$ par son équivalent inductif de type $diffelnb(l,n)$ de manière à mieux l'éliminer.

4.6 Relations dans QT liées aux variables

Pour commencer voici deux prédicats, respectivement sur $var \times QT$. et sur QT.

Définition 25 *On dit que le quasiterme t est inférieur par ses variables au quasiterme t' et on note infv(t,t') si et seulement si toute variable de t est aussi une variable de t' et s'il existe dans t' une variable qui n'est pas dans t.*

Définition 26 *On dit que le quasiterme t est clos s'il ne contient aucune variable.*

Pour manipuler les variables d'un quasiterme, on va "oublier" les constructeurs du quasiterme pour ne regarder que ses variables. Pour cela on se sert du schéma de récursion des quasitermes et on procède par filtrage sur t pour définir la liste des variables du quasiterme t.

Définition 27 *On définit récursivement l'application list-var de QT vers listv par*

1. list-var(V(x)) se simplifie en Consv(x,nil).

2. *list-var(C(f)) se simplifie en Nilv.*

3. *list-var(Root(f,t)) se simplifie en list-var(t).*

4. *list-var(ConsArg(t_1,t_2)) se simplifie en Appv(list-var(t_1),list-var(t_2)).*

On démontre ensuite les lemmes suivants.

Lemme 9

1. *Quels que soient le quasiterme t et la variable x, x est dans t si et seulement si x est dans la liste list-var(t).*

2. *Si list-var(t) contient 0 élément, alors t est clos.*

3. *ConsArg(t_1,t_2) est clos si et seulement si t_1 et t_2 sont clos.*

4. *Si t est clos, alors list-var(t) contient 0 élément.*

5. *t est inférieur par ses variables à t' si et seulement si list-var(t) est strictement incluse dans list-var(t') au sens ensembliste.*

6. *Si t est clos, alors aucun quasiterme n'est inférieur par ses variables à t.*

4.7 Prédicats sur les substitutions liés aux variables

Tout d'abord, quelques résultats sur les quasisubstitutions extensivement égales à V.

Lemme 10

1. *Si s est une quasisubstitution telle que pour toute variable x, s(x)=V(x), alors pour tout quasiterme t, Subst(s,t)=t. C'est en particulier le cas si s=$\lambda x.V(x)$.*

2. *Si t est clos, alors quelle que soit la quasisubstitution s, Subst(s,t)=t.*

On sait que dans la pratique les substitutions n'affectent qu'un nombre fini de variables. On va désigner par *domaine* de la quasisubstitution s les variables affectées par $Subst(s)$. On appelle *image* de s l'ensemble des variables des termes $s(y)$ lorsque y parcourt le domaine de s. On appelle *support* de s tout quasiterme t contenant les variables du domaine de s. On appelle *couverture* de s tout quasiterme t contenant les variables de l'image de s. Comme les sous-ensembles de E sont traduits dans le C.C. avec types inductifs par des prédicats sur E, on pose les définitions suivantes.

Définition 28 *Soit s une quasisubstitution.*

1. *On dit que la variable x est dans le domaine de s, si $s(x)\neq V(x)$. On note dom(s,x) "x est dans le domaine de s".*

2. *On dit que la variable x est dans l'image de s, s'il existe y dans le domaine de s tel x soit dans s(y). On note range(s,x) "x est dans l'image de s".*

3. *On dit que s est supportée par le quasiterme t si aucune variable n'appartenant pas à t n'est affectée par s. On note over(s,t) "s est supportée par t".*

4. *On dit que s est couverte par t si toute variable de l'image de s est dans t. On note under(s,t) "s est couverte par t".*

On obtient alors les résultats suivants concernant le domaine.

Lemme 11 *Soient s une quasisubstitution et x une variable.*

1. *Si x n'est pas dans le domaine de s, alors s(x)=V(x).*

2. *La proposition "x est dans le domaine de s" est décidable.*

Le premier résultat n'est là que pour éviter une double négation et le second pour pouvoir raisonner par cas.

Sur l'image, on démontre facilement les lemmes suivants.

Lemme 12

1. *Quelles que soit la variable x et la quasisubstitution s, si x n'est pas dans l'image de s, alors pour tout y dans le domaine de s, x n'est pas dans s(y).*

2. *Si x n'est pas dans l'image de s et si x est dans s(y), alors x=y.*

3. *Si x est dans Subst(s,t) et si x n'est pas dans l'image de s, alors x est dans t.*

Les lemmes qui suivent sont techniques, ils ont pour but d'utiliser au mieux les hypothèses de la forme "x est dans Subst(s,t)".

Lemme 13 *Soient s une quasisubstitution et t un quasiterme.*

1. *Si x est dans Subst(s,t), alors il existe y, tel que y soit dans t et x dans s(y).*

2. *Si x est dans Subst(s,t) et si s est couverte par t, alors x est dans t.*

5 Ordre sous-quasiterme dans QT

La notion de sous-terme strict est traduite par une relation binaire dans QT dont on construira deux définitions (inductive et récursive).

Définition 29 *On définit inductivement la relation sous-quasiterme strict comme étant la plus petite relation binaire dans QT vérifiant quels que soient les quasitermes t, u, et v*

1. *u est un sous-quasiterme strict de ConsArg(u,t),*

2. *u est un sous-quasiterme strict de ConsArg(t,u),*

3. *pour tout f dans fun, u est un sous-quasiterme strict de Root(f,u),*

4. *si v est un sous-quasiterme strict de t, alors v est un sous-quasiterme strict de ConsArg(t,u),*

5. *si v est un sous-quasiterme strict de t, alors v est un sous-quasiterme strict de ConsArg(u,t),*

6. *pour tout symbole f dans fun, si u est un sous-quasiterme strict de t, alors u est un sous-quasiterme strict de Root(f,t). On note sub(t,u) "t est un sous-terme strict de u".*

La version récursive s'obtient en deux temps.

Définition 30 *On définit récursivement le prédicat SUP sur QT×QT : SUP(t,u) est le terme défini par filtrage sur t tel que*

1. *SUP(x,u) se simplifie en False,*

2. *SUP(c,u) se simplifie en False,*

3. *SUP(Root(f,t),u) se simplifie en ((t=u) ou SUP(t,u)),*

4. *SUP(ConsArg(t,t'),u) se simplifie en ((t=u) ou (t'=u) ou SUP(t,u) ou SUP(t',u)).*

La version récursive de "est un sous-quasiterme strict" est notée SUB.

Définition 31 *Quels que soient les quasitermes t et u, SUB(t,u) est par définition le type SUP(u,t).*

Le passage par SUP est motivé par le fait qu'une définition est plus simple si on fait le filtrage sur le premier argument. On prouve en faisant une induction sur u que les propositions $sub(t,u)$ et $SUB(t,u)$ sont équivalentes. Les résultats importants sont les théorèmes suivants.

Théorème 5 *La relation sous-quasiterme strict dans QT est transitive.*

Démonstration. Il s'agit de prouver que pour trois quasitermes t_0, t_1 et t_2 quelconques, ($SUB(t_0,t_1)$ et $SUB(t_1,t_2)$) implique $SUB(t_0,t_2)$. Pour cela on fait une induction sur t_1, puis sur t_2, en utilisant tantôt la définition inductive de la relation sous-quasiterme, tantôt la definition récursive.

Théorème 6 *Quels que soient les quasitermes u et v, si u est un sous-quasiterme strict de v, alors $u \neq v$.*

Théorème 7 *Quels que soient la quasisubstitution s et les quasitermes u et v, si u est un sous-quasiterme strict de v, alors Subst(s,u) est un sous-quasiterme strict de Subst(s,v). Autrement dit, SUB est stable par substitution.*

Théorème 8 *Quels que soient la quasisubstitution s , la variable x et le quasiterme t. si x est dans t et si t n'est pas égal à V(x), alors s(x) est un sous-quasiterme strict de Subst(s,t).*

6 Idempotence et composition de substitutions

Définition 32 *Soit s une quasisubstitution. On dit que s est idempotente si quelle que soit la variable x, s(x) = Subst(s,s(x)).*

L'idempotence d'une quasisubstitution s est liée à son image et à son domaine.

Lemme 14 *Soit une quasisubstitution s.*

1. *Si aucune variable de l'image de s n'est dans son domaine, alors s est idempotente.*
2. *Réciproquement, si s est idempotente, alors aucune variable du domaine de s n'est dans son image.*
3. *Autrement dit, si s est idempotente, alors pour aucune variable x du domaine de s, il n'existe de variable y telle que x soit dans s(y).*

Démonstration de 1) Pour prouver que s est idempotente, il faut montrer que quelle que soit la variable x, $s(x) = Subst(s, s(x))$.
Deux cas possibles : ou bien x est dans le domaine de s, ou bien elle ne l'est pas.

a) Si x n'est pas dans le domaine de s, $V(x) = s(x)$, donc $Subst(s, s(x)) = Subst(s, V(x))$ qui se simplifie en $s(x)$.

b) Supposons que x soit dans le domaine de s. On sait (lemme 10) que, quel que soit le quasiterme t, $Subs(V, t) = t$. Il suffit donc de prouver que $Subst(s, s(x)) = Subst(V, s(x))$. Toute variable v de $s(x)$ est par définition dans l'image de s, donc, par hypothèse, n'est pas dans son domaine, et vérifie par suite $s(v) = V(v)$. Comme les variables de $s(x)$ ont même image par s et par V, on en déduit (voir lemme 3) que $Subst(s, s(x)) = Subst(V, s(x))$.

Démonstration de 2) Il faut démontrer que, si s est idempotente et si x est dans l'image de s, alors x n'est pas dans le domaine de s, c'est à dire $V(x) = s(x)$.
s étant idempotente, pour toute variable y, $s(y) = Subst(s, s(y))$.
De plus $Subst(V, s(y))$ se simplifie en $s(y)$, on a donc $Subst(V, s(y)) = Subst(s, s(y))$.
On déduit alors (lemme 3), que pour toute variable x de $s(y)$, $V(x) = s(x)$. Aucune variable de $s(y)$ n'est donc dans le domaine de s et par suite aucune variable de

l'image de s n'est dans son domaine.
Démonstration de 3) Se déduit de la démonstration précédente.

De ces lemmes on va déduire un théorème qui va jouer un rôle fondamental par la suite. Il exprime la stabilité de l'idempotence par composition.

Théorème 9 *Soient s et r deux quasisubstitutions idempotentes et t un quasiterme. Si r est supportée et couverte par Subst(s,t), alors l'application $\lambda x.Subst(r, s(x))$ est une quasisubstitution idempotente.*

Démonstration. Il faut démontrer que, si q est la quasisubstitution $\lambda x.Subst(r, s(x))$. alors q est idempotente.
D'après le *1)* du lemme il suffit donc de prouver que si x est dans l'image de q, alors il n'est pas dans son domaine. Soit x dans l'image de q. Par définition de l'image, il existe y dans le domaine de q tel que x soit dans $q(y)$ donc tel que x soit dans $Subst(r, s(y))$.
a) Supposons que x soit dans le domaine de r. Comme r est idempotente, d'après le *3)* du lemme précédent, il n'existe pas de variable y telle que x soit dans $r(y)$. x n'est donc pas non plus dans $Subst(r, s(y))$, donc pas dans l'image de q, ce qui est contradictoire. On a par conséquent $r(x) = V(x)$.
b) Supposons que x soit dans le domaine de s. s étant idempotente, x ne peut être dans aucun $s(x_0)$ en vertu de *3)* du lemme précédent. On va montrer que x est nécessairement dans $s(y)$ d'où la contradiction. D'après le lemme (voir lemme 12), x étant dans $Subst(r, s(y))$, pour montrer que x est dans $s(y)$ il suffit de montrer que x n'est pas dans l'image de r.
Supposons que x soit dans l'image de r. On a alors x dans $Subst(s, t)$ puisque $Subst(s, t)$ couvre r. Ceci implique que x est dans un $s(y_0)$ où y_0 est une variable de t, ce qui est contradictoire. x n'est donc pas dans l'image de r. x n'est par suite pas dans le domaine de s et on a $s(x) = V(x)$.
On peut alors conclure que $q(x) = Subst(r, s(x)) = Subst(r, V(x))$ qui se simplifie en $V(x)$. Cela prouve que x n'est pas dans le domaine de q. Donc aucune variable de l'image de q n'est dans son domaine. q est donc idempotente.

En utilisant un raisonnement par induction sur t, on prouve sans difficulté le résultat technique suivant.

Lemme 15 *Quelles que soient les quasisubstitutions s et r, Subst(r.Subst(s.t))=Subst(λx.Subst(r,s(x)),t).*

Le théorème final nécessite les deux théorèmes suivants qui lient ConsArg à la composition des substitutions et aux notions de support et de couverture.

Théorème 10 *Soient quatre quasitermes t_1, t_2, t_3 et t_4 et deux quasisubstitutions s et r, si s est supportée et couverte par ConsArg(t_1, t_2) et si r est supportée par ConsArg(Subst(s,t_2).Subst(s,t_4)), alors λx.Subst(r,s(x)) est supportée par ConsArg(ConsArg(t_1,t_3).ConsArg(t_2,t_4)).*

Théorème 11 *Soient quatre quasitermes t_1, t_2, t_3 et t_4 et deux quasisubstitutions s et r. si s est supportée et couverte par $ConsArg(t_1, t_2)$ et si r est suportée et couverte par $ConsArg(Subst(s,t_3),Subst(s,t_4))$, alors $\lambda x.Subst(r,s(x))$ est couverte par $ConsArg(ConsArg(t_1,t_3),ConsArg(t_2,t_4))$.*

Démonstration du support. Pour démontrer ce résultat on s'appuie sur le lemme précédemment énoncé (voir lemme 12) qui affirme que si la variable x n'est pas dans l'image de s mais est dans $Subst(s,t)$ alors x est dans t.
Il faut prouver que si x n'est pas dans $ConsArg(ConsArg(t_1,t_3),ConsArg(t_2,t_4))$, alors $V(x) = Subst(r, s(x))$.
Soit x non dans $ConsArg(ConsArg(t_1,t_3),ConsArg(t_2,t_4))$. On peut montrer que x n'est pas non plus dans $ConsArg(t_1, t_2)$ qui supporte s, et par conséquent que $V(x) = s(x)$.
La démonstration sera terminée si on peut montrer que $V(x) = r(x)$.
Si x est dans le domaine de r, x est dans $ConsArg(Subst(s,t_3),Subst(s,t_4))$ qui supporte r. x est donc par simplification dans $Subst(s,ConsArg(t_3,t_4))$. Par ailleurs x n'est pas dans $ConsArg(t_1, t_2)$ qui couvre s et par conséquent n'est pas dans l'image de s.
On peut donc appliquer le lemme 12, ce qui prouve que x est dans $ConsArg(t_3, t_4)$ et est en contradiction avec l'hypothèse
x n'est pas dans $ConsArg(ConsArg(t_1,t_3),ConsArg(t_2,t_4))$. x n'est donc pas dans le domaine de r et $V(x) = r(x)$.
Démonstration de la couverture. Pour démontrer ce résultat on s'appuie sur le lemme 13.2 qui affirme que pour que la variable x soit dans le quasiterme t il suffit que x soit dans $Subst(s,t)$ et que la quasisubstitution s soit couverte par t. On note q la quasisubstitution $\lambda x.Subst(r, s(x))$.

Il s'agit de démontrer que si x est dans l'image de la quasisubstitution q , alors x est dans $ConsArg(ConsArg(t_1,t_3),ConsArg(t_2,t_4))$.
Comme $q(y)$ se réduit en $Subst(r, s(y))$ cela revient à prouver que tout x,
pour lequel il existe y dans le domaine de q tel que x soit dans $Subst(r, s(y))$,
est dans $ConsArg(ConsArg(t_1,t_3),ConsArg(t_2,t_4))$.

L'utilisation du lemme 13.2 réduit ce but à deux sous buts :
a) r est couverte par $ConsArg(ConsArg(t_1,t_3),ConsArg(t_2,t_4))$ et
b) x est dans $Subst(r,ConsArg(ConsArg(t_1,t_3),ConsArg(t_2,t_4)))$.

Démonstration de a) Soit u dans l'image de r. u est dans $ConsArg(Subst(s,t_3), Subst(s,t_4))$ qui couvre r, donc aussi dans $Subst(s, ConsArg(t_3, t_4))$. Il existe donc v dans $ConsArg(t_3, t_4)$, tel que u soit dans $s(v)$.

Si v n'est pas dans le domaine de s, $s(v) = V(v)$ et par conséquent $u = v$, donc u est dans $ConsArg(t_3, t_4)$, donc dans $ConsArg(ConsArg(t_1, t_3), ConsArg(t_2, t_4))$.

Si v est dans le domaine de s, u est dans dans l'image de s donc dans $ConsArg(t_1, t_2)$ qui couvre s, donc dans $ConsArg(ConsArg(t_1, t_3), ConsArg(t_2, t_4))$.

Démonstration de b) On sait que x est dans $r(v)$ avec v dans $s(y)$. Il est impossible que y ne soit ni dans le domaine de s ni dans celui de r car dans ce

cas, contrairement aux hypothèses, il ne serait pas non plus dans celui de $q = \lambda x.Subst(r, s(x))$.

Si y n'est pas dans le domaine de s, il est nécessairement dans celui de r. On alors $s(y) = V(y)$ et y dans $Subst(s, ConsArg(t_3, t_4))$ qui couvre r. On en déduit que y est dans $Subst(s, ConsArg(ConsArg(t_1,t_3), ConsArg(t_2,t_4)))$. Or s est couverte par $ConsArg(t_1,t_2)$ donc par $ConsArg(ConsArg(t_1,t_3), ConsArg(t_2,t_4))$ et l'application du lemme 13.2 permet d'affirmer que y est dans $ConsArg(ConsArg(t_1,t_3).ConsArg(t_2,t_4))$. On en déduit que, dans ce cas, x qui est dans $r(y)$
est dans $Subst(r, ConsArg(ConsArg(t_1,t_3), ConsArg(t_2,t_4)))$

Si y est dans le domaine de s, y est dans $ConsArg(t_1,t_2)$ qui supporte s et x qui est dans $r(y)$ est dans $Subst(r, ConsArg(t_1,t_2))$ donc dans $Subst(r, ConsArg(ConsArg(t_1,t_3).ConsArg(t_2,t_4)))$.

La démonstration du théorème qui suit montre la nécessité du support.

Théorème 12 *Si une quasisubstitution est supportée par un quasiterme t, ou bien il existe une variable x telle que $V(x) \neq s(x)$, ou bien pour toute variable x, $V(x) = s(x)$.*

Démonstration. On commence par démontrer le résultat dans le cas où on peut mettre toutes les variables du domaine de s dans une liste. On fait pour cela un raisonnement par induction sur cette liste. On applique ensuite ce résultat à la liste des variables de t (list-var(t)).

Tout comme les autres résultats de décidabilité, ce théorème servira dans des raisonnements par cas.

6.1 Idempotence et diminution du nombre de variables distinctes

Le théorème qui suit va jouer un rôle primordial dans la preuve de terminaison de l'algorithme d'unification.

Théorème 13 *Soient quatre quasitermes t_1, t_2, t_3 et t_4, s une quasisubstitution, n et n_0 deux entiers naturels. Si*
a) s est idempotente,
b) la liste des variables du quasiterme $ConsArg(ConsArg(t_1,t_2).ConsArg(t_3,t_4))$ contient n variables distinctes,
c) s est supportée et couverte par $ConsArg(t_1,t_3)$,
d) le domaine de s contient au moins la variable x.
e) la liste des variables de $ConsArg(Subst(s,t_2), Subst(s,t_4))$ contient n_0 variables distinctes,
alors $n_0 < n$.

Démonstration. Il suffit de démontrer que la liste des variables de *ConsArg(Subst(s,t2),Subst(s,t4))* est strictement incluse dans la liste des variables de *ConsArg(ConsArg(t1,t2),ConsArg(t3,t4))* (voir *DIFFELNB_influe_S*).
Soit x une variable du domaine de s. x apparait dans la seconde liste et pas dans la première. En effet on a démontré (voir lemme 14.3) que si s est idempotente aucune variable de son domaine n'apparait dans un quasiterme de la forme $s(y)$, donc si x était dans la première liste, x serait également dans *ConsArg(Subst(s,t2),Subst(s,t4))* et il existerait x_0 telle que $s(x_0)$ contienne x, ce qui est impossible.
Si une variable y est dans la première liste, elle est aussi dans la seconde. Si y est dans la première liste, y est dans *ConsArg(Subst(s,t2),Subst(s,t4))* et il existe une variable u dans *ConsArg(t2,t4)* tel que y soit dans $s(u)$.

Si u est dans le domaine de s, y est dans l'image de s qui est couverte par *ConsArg(t1, t3)*, y est donc aussi dans *ConsArg(ConsArg(t1,t2),ConsArg(t3,t4))*.

Si u n'est pas dans le domaine de s, $V(u) = s(u)$ et $y = u$ puisque y est dans $s(u)$. y est dans *ConsArg(t2,t4)*, donc y est dans *ConsArg(ConsArg(t1,t2),ConsArg(t3,t4))*.

6.2 Construction itérative de substitutions

On a choisi *nat* comme ensemble de variables. On va explicitement se servir de ce choix pour construire des quasisubstitutions de manière itérative. En effet, pour modifier une quasisubstitution en une variable x, il faut pouvoir définir itérativement la nouvelle quasisubstitution.

Théorème 14 Modification ponctuelle d'une quasisubstitution.
Si var ≡ nat, alors étant donnés une quasisubstitution s, un quasiterme t et une variable x, il existe une quasisubstitution r qui vérifie

1. $r(x) = t$,
2. *si y est une variable distincte de x, alors* $r(y) = s(y)$.

Démonstration. On fait ici un raisonnement par induction sur l'entier x.
Pour $x = 0$, on construit r_0 récursivement en définissant $r_0(n)$ par filtrage sur n. $r_0(0)$ se simplifiant en t et $r_0(n+1)$ en $s(n+1)$.
Pour $x = n+1$, on applique l'hypothèse de récurrence à $\lambda p.s(p+1)$. Cela prouve qu'il existe une quasisubstitution q_n valant t en n et $s(p+1)$ en p pour $p \neq n$. r_{n+1} est défini récursivement, $r_{n+1}(0)$ se simplifiant en $s(0)$ et $r_{n+1}(p+1)$ en $q_n(p)$ donc en $s(p+1)$.

On en déduit le théorème suivant.

Théorème 15 Quasisubstitution identité sauf en un point. *Etant donné un quasiterme t et une variable x, il existe une quasisubstitution r qui vérifie*

1. $r(x)=t$.

2. *si y est une variable distincte de x, alors r(y)=V(y).*

On dira alors que r est une quasisubstitution élémentaire et on notera $elem - subst(x,t,r)$ cette proposition.
Une telle quasisubstitution vérifie bien entendu le théorème suivant.

Théorème 16 *Si le domaine de la quasisubstitution s est réduit à x et si la variable x n'est pas dans le quasiterme t, alors Subst(s,t)=t.*

Démonstration. s et V ont même restriction sur l'ensemble des variables de t, on en déduit que $Subst(s,t) = Subst(V,t)$. Or $Subst(V,t)$ se simplifie en t donc $Subst(s.t) = t$.

7 Unification

7.1 Unificateur minimal

On commence par définir inductivement dans l'ensemble des quasisubstitutions une relation notée $less - subst.\ less - subst(s,r)$ se lira "s est inférieure à r".

Définition 33 *La relation less-subst est la plus petite relation binaire sur l'ensemble des quasisubstitutions vérifiant l'assertion suivante :*
quelles que soient les quasisubstitutions s et r, s'il existe une quasisubstitution q telle que, pour toute variable x, Subst(q,s(x))=r(x), alors less-subst(s,r).

Définition 34 *Etant donnés deux quasitermes t et u et une quasisubstitution s, on dit que s unifie t et u, si Subst(s,t)=Subst(s,u).*

Définition 35 *Etant donnés deux quasitermes t et u et une quasisubstitution s, on dit que s est minimal par rapport à t et u, si s est inférieur à tout unificateur de t et u.*

Si s unifie t et u et est minimal par rapport à t et u, on dit que s est un unificateur minimal de t et u. On peut remarquer que si t et u n'admettent pas d'unificateur, toute quasisubstitution est minimale par rapport à t et u, mais cela n'a aucune importance.

7.2 Spécification du problème de l'unification

Définition 36 *On définit inductivement la résolution du problème de l'unification comme la plus petite spécification vérifiant, quels que soient les deux quasitermes t et u, les deux assertions suivantes :*

1. *S'il existe une quasisubstitution s idempotente, unificateur minimal de t et u, supportée et couverte par Cons-Arg(t.u), alors le problème de l'unification de t et u est résolu. (On dit que l'unification réussit)*

2. *Si, quelle que soit la quasisubstitution s, $Subst(s,t) \neq Subst(s,u)$, alors le problème de l'unification de t et u est résolu.(On dit que l'unification échoue)*

7.3 Cas d'échec à cause du constructeur de tête

Définition 37 *On définit inductivement il y a échec par la tête pour deux quasitermes comme la plus petite relation sur $QT \times QT$ vérifiant les assertions suivantes :*

1. *il y a échec par la tête pour C(f) et Root(f',t),*
2. *il y a échec par la tête pour Root(f',t) et C(f),*
3. *il y a échec par la tête pour C(f) et ConsArg(t,u).*
4. *il y a échec par la tête pour ConsArg(t,u) et C(f),*
5. *il y a échec par la tête pour Root(f',t) et ConsArg(t,u),*
6. *il y a échec par la tête pour ConsArg(t,u) et Root(f',t).*

On obtient alors le premier résultat sur l'unification.

Théorème 17 *S'il y a échec par la tête pour les deux quasitermes t et u, alors le problème de l'unification de t et u est résolu.*

Démonstration. S'il y a échec par la tête pour t et u, pour toute quasisubstitution s, $Subst(s,t)$ et t s'écrivent avec le même symbole de tête dans le C.C. avec types inductifs ainsi que $Subst(s,u)$ et u. $Subst(s,t)$ et $Subst(s,u)$ sont donc toujours distincts et l'unification échoue.

La symétrie de l'égalité permet de démontrer le lemme suivant.

Lemme 16 *Quels que soient les deux quasitermes t et u, si le problème de l'unification de t et u est résolu, alors le problème de l'unification de u et t est résolu.*

7.4 Cas où l'un des quasitermes est une variable

Théorème 18 *Quels que soient le quasiterme t et la variable x, le problème de l'unification de t et de V(x) et celui de V(x) et de t sont résolus.*

Démonstration. On va faire un raisonnement par cas.

1. On suppose que x est dans t.
   (a) On suppose que $V(x) = t$. Dans ce cas l'unification réussit avec la quasisubstitution V puisque $Subst(V, V(x))$ se simplifie en $V(x)$ qui est égal à t.

(b) On suppose que $V(x) \neq t$. Comme on l'a démontré antérieurement pour toute quasisubstitution s, si x est dans t, $Subst(s, x)$ est un sous-quasiterme strict de $Subst(s, t)$ et on sait qu'alors $Subst(s, x)$ et $Subst(s, t)$ sont différents. Il y dans ce cas échec à l'unification.

2. On suppose que x n'est pas dans t. On sait qu'il existe une quasisubstitution élémentaire r qui vérifie $r(x) = t$ et pour toute variable y telle que $y \neq x$, $r(y) = V(y)$. Comme x n'est pas dans t, $Subst(r, t) = t$ et, comme par définition de r, $Subst(r, V(x))$ se simplifie en t, r unifie t et $V(x)$.
Pour montrer que r est idempotente, il faut montrer que pour toute variable x_0, $Subst(r, r(x_0)) = r(x_0)$. On a soit $x_0 = x$, soit $x_0 \neq x$.

   Si $x_0 = x$, $r(x_0) = t$ et $Subst(r, r(x_0)) = Subst(r, t) = t$, puisque la seule variable du domaine de r est x qui n'est pas dans t.

   Si $x_0 \neq x$, $r(x_0) = V(x_0)$ et $Subst(r, r(x_0)) = Subst(r, V(x_0))$ qui se simplifie en $r(x_0)$ qui est égal par définition à $V(x_0)$.
   Le domaine de r ne contient que x, r est supportée par $ConsArg(V(x), t)$.
   Les variables de l'image de r sont les variables de t, r est donc couverte par $ConsArg(V(x), t)$.
   Si q unifie $V(x)$ et t, on a $Subst(q, V(x)) = Subst(q, t)$. On a donc, comme $Subst(q, V(x))$ se simplifie en $q(x)$ et comme $t = r(x)$, $q(x) = Subst(q, r(x))$.
   D'autre part si $x_0 \neq x$, $Subst(q, r(x_0)) = Subst(q, V(x_0))$, qui se simplifie en $q(x_0)$. On a donc $Subst(q, r(x_0)) = q(x_0)$. r est donc inférieur à q. r est donc minimal.

On a ainsi prouvé la résolution du problème de l'unification de t et de $V(x)$. Par symétrie, on en déduit la résolution du problème de l'unification de $V(x)$ et de t.

7.5 Cas où un des quasitermes est une constante

Théorème 19 *Quels que soient le quasiterme t et la constante f, le problème de l'unification de t et $C(f)$ ainsi que celui de $C(f)$ et t sont résolus.*

Démonstration. On fait un raisonnement par induction sur t. Les théorèmes précédents permettent de conclure quand t est un quasiterme variable, et quand t admet *Root* ou *ConsArg* comme constructeur de tête. Il reste le cas où t est un terme constant $C(f_0)$. Il y a alors deux possibilités : ou bien $f_0 = f$, ou bien $f_0 \neq f$.

Si $f_0 = f$, l'unification réussit avec V. V est idempotente, le domaine de V étant vide, V est supportée et couverte par $ConsArg(C(f_0), C(f))$ et comme pour toute quasisubstitution s, $Subst(s, V(x))$ se simplifie en $s(x)$, V est un unificateur minimal de $C(f_0)$ et $C(f)$.

Si $f_0 \neq f$, quelle que soit la quasisubstitution s, $Subst(s, C(f_0))$ se simplifie en $C(f_0)$ et $Subst(C(f))$ en $C(f)$, l'unification échoue donc.

7.6 Le constructeur de tête des deux quasitermes est Root

Théorème 20 *Soient f_0 et f_1 deux éléments de fun et deux quasitermes t_0 et t_1. Si le problème de l'unification est résolu pour t_0 et t_1, alors il l'est pour $Root(f_0,t_0)$ et $Root(f_1,t_1)$.*

Démonstration. Si $f_0 = f_1$ et si l'unification de t_0 et t_1 réussit avec s comme unificateur minimal on vérifie que s convient encore pour $Root(f_0, t_0)$ et $Root(f_1, t_1)$. Le seul problème qui se pose est que s soit minimal. On remarque que tout unificateur r de $Root(f_0, t_0)$ et de $Root(f_1, t_1)$ en est encore un pour t_0 et t_1 et est donc supérieur à s. Cela se fait en utilisant un lemme de projection.

Si l'unification de t_0 et t_1 échoue, quelle que soit la quasisubstitution s, $Subst(s, Root(f_0, t_0))$ se simplifie en $Root(f_0, Subst(s, t_0))$, et $Subst(s, Root(f_1, t_1))$ en $Root(f_1, Subst(s, t_1))$. On conclut en utilisant un lemme de non-égalité.

Si $f_0 \neq f_1$, on remarque en que quelle que soit la quasisubstitution s, $Subst(s, Root(f_0, t_0))$ se simplifie en $Root(f_0, Subst(s, t_0))$ et $Subst(s, Root(f_1, t_0))$ en $Root(f_1, Subst(s, t_0))$, il y a donc échec de l'unification. On utilise là encore un lemme de non-égalité.

7.7 Le constructeur de tête des deux quasitermes est ConsArg

Lemme 17 *Soient t_1, t_2, t_3 et t_4 quatre quasitermes. Si l'unification de t_1 avec t_3 échoue, alors il en est de même pour celle de $ConsArg(t_1,t_2)$ avec $ConsArg(t_3,t_4)$.*

Démonstration. C'est un résultat évident qui d'ailleurs ne sera utilisé qu'avec l'échec de l'unification de t_1 avec t_3. On le démontre en utilisant des lemmes de projection. Le cas où l'unification de t_1 et de t_3 réussit constitue le coeur du problème de l'unification et est étudié plus loin.

7.8 Les deux quasitermes sont clos

Lemme 18 *Soient t_1 et t_2 deux quasitermes. Si la liste des variables de $ConsArg(t_1,t_2)$ contient 0 variable, le problème de l'unification de t_1 avec t_2 est résolu.*

Démonstration. Si $t_1 = t_2$, on montre que l'unification réussit avec V. Sinon elle échoue. En effet, si la liste des variables de $ConsArg(t_1, t_2)$ contient 0 variable, $ConsArg(t_1, t_2)$ donc t_1 et t_2 sont clos, donc pour toute quasisubstitution s, $Subst(s, t_1) = t_1$ et $Subst(s, t_2) = t_2$, donc $Subst(s, t_1) \neq Subst(s, t_2)$.

7.9 Lemmes techniques sur le nombre de variables distinctes

Lemme 19 *Soient t_0, t_1, u_0, u_1 quatre quasitermes et n et p deux entiers. Si $ConsArg(ConsArg(t_0,t_1),ConsArg(u_0,u_1))$ contient n variables distinctes et si*

$ConsArg(t_0.u_0)$ contient p variables distinctes, alors $p < n$ ou bien $p = n$.
Si $ConsArg(ConsArg(t_0.t_1),ConsArg(u_0,u_1))$ contient n variables distinctes et si $ConsArg(t_1,u_1)$ contient p variables distinctes, alors $p < n$ ou bien $p = n$.

Démonstration. La démonstration utilise le fait déjà démontré que si une liste l_0 est incluse dans une liste l, l contient au moins autant de variables distinctes que l_0.

7.10 Composition et unificateur minimal

Lemme 20 *Soient t_0, t_1, u_0, u_1 quatre quasitermes et s et r deux quasisubstitutions. Si s unifie t_0 et u_0 et si r unifie $Subst(s,t_1)$ et $Subst(s,u_1)$, alors $\lambda x.Subst(r,s(x))$ unifie $ConsArg(t_0,t_1)$ et $ConsArg(u_0,u_1)$.*

Démonstration. On utilise le fait que $Subst(r, Subst(s,t)) = Subst(\lambda x.Subst(r,s(x)),t)$.

Lemme 21 *Soient t_0, t_1, u_0, u_1 quatre quasitermes et s et r deux quasisubstitutions. Si s est minimal par rapport à t_0 et u_0 et r est minimal par rapport à $Subst(s,t_1)$ et $Subst(s,u_1)$, alors $\lambda x.Subst(r,s(x))$ est minimal par rapport à $ConsArg(t_0.t_1)$ et $ConsArg(u_0,u_1)$.*

Démonstration. Si s_0 unifie $ConsArg(t_0,t_1)$ et $ConsArg(u_0,u_1)$, alors s_0 unifie t_0 et u_0, ainsi que t_1 et u_1.
Il existe donc une quasisubstitution q telle que, pour toute variable x, $s_0(x) = Subst(q,s(x))$.
De l'égalité $Subst(s_0,t_1) = Subst(s_0,u_1)$,
on déduit que $Subst(\lambda x.Subst(q,s(x)),t_1) = Subst(\lambda x.Subst(q,s(x)),u_1)$
donc que $Subst(q,Subst(s,t_1)) = Subst(q,Subst(s,u_1))$ ce qui prouve que q unifie $Subst(s,t_1)$ et $Subst(s,u_1)$. On en déduit que r est inférieure à q, il existe donc une quasisubstitution q_0 telle que pour toute variable x, $q(x) = Subst(q_0,r(x))$.
Pour toute variable x, $s_0(x) = Subst(q,s(x))$ par définition de q,
donc $s_0(x) = Subst(\lambda x.Subst(q_0,r(x)),s(x))$ par définition de q_0,
donc $s_0(x) = Subst(q_0,Subst(r,s(x)))$, donc $s_0(x) = Subst(q_0,\lambda x.Subst(r,s(x)),x)$.
La quasisubstitution $\lambda x.Subst(r,s(x)),x)$ est donc inférieur à s_0. $\lambda x.Subst(r,s(x)),x)$ est donc minimale par rapprt à $ConsArg(t_0,t_1)$ et $ConsArg(u_0,u_1)$.

Lemme 22 *Soient t_0, t_1, u_0, u_1 quatre quasitermes et s et r deux quasisubstitutions. Si s est un unificateur minimal de t_0 et u_0 idempotent supporté et couvert par $ConsArg(t_0,u_0)$ et si r est un unificateur minimal de $Subst(s,t_1)$ et $Subst(s,u_1)$ idempotent supporté et couvert par $ConsArg(Subst(s,t_1),Subst(s,u_1))$, alors l'unification de $ConsArg(t_0.t_1)$ et de $ConsArg(u_0.u_1)$ réussit.*

Démonstration. La seule chose qui n'est pas une conséquence des lemmes précédents est l'idempotence de $\lambda x.Subst(r,s(x))$ qui a été démontrée dans un des paragraphe précédent.

Lemme 23 *Soient t_0, t_1, u_0, u_1 quatre quasitermes et une quasisubstitution s. Si s est un unificateur minimal de t_0 et u_0 idempotent supporté et couvert par ConsArg(t_0,u_0) et si l'unification de Subst(s,t_1) et Subst(s,u_1) échoue, alors il en est de même pour celle de ConsArg(t_0,u_0) et ConsArg1(t_1,u_1).*

Démonstration. S'il en était autrement $Subst(s, t_1)$ et $Subst(s, u_1)$ seraient unifiables à cause de la minimalité de s.

7.11 Unification de ConsArg(t,u) et de ConsArg(t,v)

Il faut traiter ce cas à part, car l'unificateur minimal de t, idempotent, est V et c'est le seul cas où il n'y a pas diminution du nombre de variables distinctes dans l'image. Cependant, ce cas apparait dans la démonstration du théorème final sous la forme s est un unificateur t_0 et u_0 tel que pour toute variable x, $V(x) = s(x)$.

Lemme 24 *Soient t_0, t_1, u_0, u_1 quatre quasitermes et s un unificateur de t_0 et u_0. Si, pour toute variable x, V(x)=s(x) et si l'unification de t_1 et de u_1 réussit, alors celle de $ConsArg(t_0, t_1)$ et $ConsArg(u_0, u_1)$ réussit également et si l'unification de t_1 et de u_1 échoue il en est de même pour celle de $ConsArg(t_0, t_1)$ et $ConsArg(u_0, u_1)$.*

Démonstration. Il suffit de vérifier qu'un unificateur minimal idempotent de t_1 et u_1 couvert et supporté par $ConsArg(t_1, u_1)$ a les mêmes propriétés vis à vis de $ConsArg(t_0, t_1)$ et $ConsArg(u_0, u_1)$.

7.12 Théorème final

Théorème 21 *Quels que soient les deux quasitermes t et u, le problème de l'unification de t et de u est résolu.*

Démonstration. Soit n un entier tel que $liste\_var(ConsArg(t, u))$ contienne n variables distinctes. La démonstration repose en premier lieu sur un raisonnement par induction sur n.
Si $n = 0$, le problème a été résolu (unification des quasitermes clos). Il reste à démontrer que si le problème est résolu au rang p pour tout entier p tel que $p \leq n$, il est résolu au rang $n+1$. On raisonne alors par induction sur t, puis par induction sur u.
Compte-tenu des théorèmes précédents il ne reste plus à étudier que le cas où t est de la forme $ConsArg(t_0, t_1)$ et u de la forme $ConsArg(u_0, u_1)$. On dispose parmi les hypothèses, en outre, de trois hypothèses d'induction qui sont :

(H_1) : Quels que soient les quasitermes p_0 et p_1, si $liste\_var(ConsArg(p_0, p_1))$ contient q variables distinctes avec $q \leq n$, alors le problème de l'unification de p_0 et p_1 est résolu.

(H_2) : Quel que soit le quasiterme w, si $liste\_var(ConsArg(t_0, w))$ contient $n+1$ variables distinctes, alors le problème de l'unification de t_0 et w est résolu.

(H_3) : Quel que soit le quasiterme w, si $liste\_var(ConsArg(t_1, w))$ contient $n+1$ variables distinctes, alors le problème de l'unification de t_1 et w est résolu.

(H_1) vient de l'induction sur n, les deux autres de l'induction sur t.
Si $liste\_var(ConsArg(t_0, u_0))$ contient n_0 variables distinctes, on soit $n_0 < n+1$ soit $n_0 = n+1$.

1. Supposons que $n_0 < n+1$. Le problème de l'unification de t_0 et u_0 est résolu d'après (H_1).
   Si l'unification de t_0 et u_0 échoue, on sait qu'il en est de même pour celle de $ConsArg(t_0, t_1)$ et $ConsArg(u_0, u_1)$.
   Si l'unification de t_0 et de u_0 réussit avec s comme unificateur, alors deux cas peuvent se présenter, ou bien le domaine de s est vide ou bien il ne l'est pas.

   (a) Si le domaine de s n'est pas vide, $liste-var(ConsArg(Subst(s, t_1), Subst(s, u_1)))$ contient m variables distinctes, avec $m < n+1$ en vertu du théorème 13.
   Le problème de l'unification de $Subst(s, t_1)$ et $Subst(s, u_1)$ est résolu en appliquant (H_1).
   Si l'unification de $Subst(s, t_1)$ et $Subst(s, u_1)$ réussit avec s_0 comme unificateur, on sait que celle de $ConsArg(t_0, t_1)$ et $ConsArg(u_0, u_1)$ réussit avec $\lambda x.Subst(s_0, s(x))$ comme unificateur (lemme 22).
   Si l'unification de $Subst(s, t_1)$ et $Subst(s, u_1)$ échoue, on sait que celle de $ConsArg(t_0, t_1)$ et $ConsArg(u_0, u_1)$ échoue également (lemme 23).

   (b) Si le domaine de s est vide, on a alors $Subst(s, t_1) = t_1$ et $Subst(s, u_1) = u_1$. Il suffit donc de prouver que le problème de l'unification de t_1 et u_1 est résolu.
   Si $liste\_var(ConsArg(t_1, u_1))$ contient m_0 variables distinctes, on sait alors que $m_0 < n+1$ ou bien $m_0 = n+1$.

      i. Si $m_0 < n+1$, on peut appliquer l'hypothèse de récurrence (H_1) à t_1 et à u_1 et conclure.

      ii. Si $m_0 = n+1$, c'est l'hypothèse de récurrence (H_3) qu'il faut appliquer à u_1 pour pouvoir conclure.

2. Supposons que $n_0 = n+1$. Il faut montrer que le problème de l'unification de $ConsArg(t_0, t_1)$ et $ConsArg(u_0, u_1)$ est résolu. C'est l'hypothèse de récurrence (H_2) qui permet d'affirmer que le problème de l'unification de t_0 et u_0 est résolu.
   La suite de la démonstration est rigoureusement la même que dans a).

8 Conservation du résultat pour les termes

La fonction *arity* est définie de *fun* vers *nat*. Comme *fun* et *nat* sont en fait le même type, *arity* est aussi une fonction de *nat* vers *nat*. En général, les symboles

fonctionnels c'est à dire les éléments de *fun* dont l'arité n'est pas nulle sont en nombre fini. *arity(f)* est donc nul pour f "assez grand". On peut donc associer à *arity* une liste d'entiers telle que *arity(f)* ait pour valeur le $(f+1)$-ième élément de la liste (l'ordre étant celui d'entrée). En pratique dans le C.C., cela se traduit par la donnée d'une liste d'éléments de *nat* nommée *list-arity*, de sa transformation en file par l'application d'un opérateur nommé *List-queue-nat*, puis de l'application à *List-queue-nat(arity)* de l'opérateur *Constr-nat* dont le résultat est une fonction de *fun* vers *nat*. *List-queue-fun* et *Constr-nat* sont définis récursivement et leurs définitions sont vérifiées. Plus précisément on pose d'abord :

Définition 38 *On définit récursivement l'application At-last de nat × list-nat vers list-nat par*

1. *At-last(n,Nil-nat) se simplifie en Cons-nat(n,Nil-nat),*
2. *At-last(n,Cons-nat(p,l)) se simplifie Cons-nat(p,At-last(n,l)).*

Définition 39 *On définit récursivement l'application List-queue-nat de list-nat vers list-nat par*

1. *List-queue-nat(Nil-nat) se simplifie en Nil-nat,*
2. *List-queue-nat(Cons-nat(p,l)) se simplifie At-last(p,List-queue-nat(l)).*

Définition 40 *On définit récursivement l'application Constr-f de list-nat vers (nat → nat) par*

1. *Constr-f(Nil-nat) se simplifie en λx :nat.O,*
2. *Constr-f(Cons-nat(p,l)) se simplifie en la fonction f définie récursivement par*
   (a) *f(O) se simplifie en p et*
   (b) *f(x+1) en Constr-f(l)(x).*

Pour traduire les notions de liste-termes et de termes on suit la présentation donnée dans l'introduction. C'est à dire que l'on définit la longueur d'un quasiterme et que l'on traduit les prédicats simple, liste-terme et terme. Cela se fait de la manière suivante :

Définition 41 *On définit récursivement le prédicat SIMPLE sur QT par*

1. *SIMPLE(V(x)) se simplifie en True,*
2. *SIMPLE(C(f)) se simplifie en True,*
3. *SIMPLE(Root(f,t)) se simplifie en True.*
4. *SIMPLE(ConsArg(t,u)) se simplifie en False.*

Définition 42 *On définit récursivement la fonction LENGTH de QT vers nat par*

1. *LENGTH(V(x)) se simplifie en 1 (noté S(O)),*
2. *LENGTH(C(f)) se simplifie en 1,*
3. *LENGTH(Root(f,t)) se simplifie en 1,*
4. *LENGTH(ConsArg(t,u)) se simplifie en LENGTH(t)+LENGTH(u) (l'addition étant redéfinie à cette occasion).*

Définition 43 *On définit récursivement le prédicat L-TERM sur QT par*

1. *L-TERM(V(x)) se simplifie en True,*
2. *L-TERM(C(f)) se simplifie en (arity(f)=O),*
3. *L-TERM(Root(f,t)) se simplifie en (L-TERM(t) et arity(f)=LENGTH(t)),*
4. *L-TERM(ConsArg(t,u)) se simplifie en (SIMPLE(t) et L-TERM(t) et L-TERM(u)).*

Définition 44 *On définit inductivement le prédicat term sur QT par*
term(t) est vrai si et seulement si (SIMPLE(t) et L-TERM(t)).

On définit la version inductive $l-term$ de $L-TERM$, puis on prouve l'équivalence des deux définitions, bien que dans ce cas la version récursive soit très claire. Les décidabilités de SIMPLE, $L-TERM$ et term sont faciles à établir.

Viennent ensuite un certain nombre de lemmes qui seront importants par la suite.

Lemme 25

1. *Quel que soit t, il existe un entier x tel que* $LENGTH(t) = x + 1$.
2. *Quel que soit t et u,* $LENGTH(ConsArg(t,u)) \neq 1$.
3. *Quel que soit t, SIMPLE(t) implique* $LENGTH(t) = 1$.
4. *Quel que soit t, si L-TERM(t) et* $LENGTH(t) = 1$, *alors term(t).*
5. *Quel que soit t, si term(t) alors L-TERM(t) et* $LENGTH(t) = 1$.

En fait, les prédicats $L-TERM$ et *term* dépendent de la donnée de $list-arity$. Dans la suite, on a fait une abstraction sur cette liste et à cause de cela $L-TERM$ et *term* dépendront non plus de un mais de deux paramètres, un de type $list-nat$ et l'autre de type QT.

On se préoccupe à nouveau des qualités *term* et $L-TERM$ lors de l'étude des substitutions. On montre en particulier que $L-TERM$ et *term* sont des propriétés conservées par la famille des substitutions qui n'ont que des termes dans leur ensemble image.

Définition 45 *On dit qu'une quasisubstitution f est compatible avec les termes définis par la liste l si quelle que soit la variable x, f(x) est un terme. On note term-subst(l,t) cette propriété.*

On a alors

Lemme 26 *quels que soient la liste l dans list-nat, la quasisubstitution f et le quasiterme t.*

1. *si term-subst(l,f), alors Length(Subst(f,t))=Length(t),*
2. *si term-subst(l,f) et SIMPLE(t), alors SIMPLE(Subst(f,t)),*
3. *si L-TERM(l,t) et term-subst(l,f), alors L-TERM(l,Subst(f,t)),*
4. *si term(l,t) et term-subst(l,f), alors term(l,Subst(f,t)).*

Enfin, c'est au moment de la spécification du problème de l'unification que se pose le problème de la conservation. Il faut modifier la définition même du problème de l'unification (nommé simplement Unification dans le listing) en imposant à l'unificateur de t_1 et de t_2 d'être une quasisubstitution compatible avec les termes chaque fois que t_1 et t_2 sont des liste-termes de même longueur. Au passage, il est intéressant de constater qu'imposer à l'unificateur de t_1 et de t_2 d'être une quasisubstitution compatible avec les termes chaque fois que t_1 *et* t_2 *sont des termes*, ne permet pas d'aboutir. Cela vient du fait que tous les raisonnements ou presque sont des raisonnements par induction où les quasitermes interviennent en tant que listes. Le problème de la conservation ne se pose évidemment qu'en cas de succès. Lorsque l'un des termes est une variable ou une constante, il suffit d'appliquer les définitions et les lemmes ci-dessus. Le seul résultat un peu compliqué à poser plutôt qu'à démontrer est le suivant.

Lemme 27 *Soient 4 quasitermes t_1, t_2, t_3 et t_4, 2 quasisubstitutions f et g et l une liste d'entiers. Si, pour la liste l d'arités,*

1. *ConsArg(t_1,t_2) et ConsArg(t_3,t_4) sont des liste-termes de même longueur,*
2. *le fait que t_1 et t_3 soient des liste-termes de même longueur implique que f est compatible avec les termes,*
3. *le fait que Subst(f,t_2) et Subst(f,t_4) soient des liste-termes de même longueur implique que g est compatible avec les termes.*

alors la substitution composée de f par g, λx :nat.Subst(g,f(x)) est compatible avec les termes.

Pour démontrer que $\lambda x : nat.Subst(g,f(x))$ est compatible avec les termes il faut prouver que pour toute variable x, $Subst(g,f(x))$ est un terme.
Supposons que f soit compatible avec les termes. On sait alors que $f(x)$ est un terme. Pour que $Subst(g, f(x))$ en soit un aussi il suffit que g soit compatible avec les termes d'après l'assertion 4) du lemme précédent.
Pour cela, il suffit que $Subst(f, t_2)$ et $Subst(f, t_4)$ soient des liste-termes de même longueur, ce qui revient à prouver, compte-tenu de l'hypothèse faite sur f, que t_2 et t_4 sont des liste-termes de même longueur.
Or, on sait que $ConsArg(t_1, t_2)$ et $ConsArg(t_3, t_4)$ sont des liste-termes de même longueur, ce qui entraine
1) que t_1 et t_3 sont des liste-termes simples donc de longueur 1,
2) que t_2 et t_4 sont des liste-termes.
On a $Length(ConsArg(t_1, t_2)) = Length(t_2) + 1$ et $Length(ConsArg(t_3, t_4)) = Length(t_4) + 1$,
or $Length(ConsArg(t_1, t_2)) = Length(ConsArg(t_3, t_4))$, donc $Length(t_2) = Length(t_4)$.
g est donc bien compatible avec les termes.
Il reste à prouver que f l'est, ce qui se déduit du fait que t_1 et t_3 sont des liste-termes de même longueur, puisque tous les deux de longueur 1.

Le lemme ci-dessus est bien sûr utilisé dans la preuve de l'unification de $ConsArg(t_1, t_2)$ et $ConsArg(t_3, t_4)$ lorsque l'on sait que l'unification de t_1 et de t_3 réussit avec f et que celle de $Subst(f, t_2)$ et $Subst(f, t_4)$ réussit avec g.

9 Quelques leçons tirées de l'expérience

La preuve qui précède a d'abord été faite en utilisant l'implantation du Calcul des Constructions (sans types inductifs) version 4.10. Le mécanisme d'extraction programmé dans le Coq Proof Assistant version 5.6 et celui de la version 4.10 sont basés sur la même notion de réalisabilité. Il y a deux différences essentielles entre les deux versions. La première est que l'on gagne de nouvelles possibilités de construction d'objets en remplaçant le type *Data* par le type *Set* (au niveau des types dépendants), la seconde est que la version 5.6 fabrique automatiquement lors de la déclaration d'un type inductif des opérateurs de récursion et les axiomes d'induction associés au type créé. Il fallait dans la version 4.10 définir les constructeurs des types après les types ainsi que les opérateurs de récursion et les axiomes d'induction associés. C'était un travail long et un peu trop répétitif, mais non dénué d'intérêt didactique. La commande *InductiveDefinition* permettait toutefois d'obtenir en posant une seule définition un type dans *Data* ou dans *Prop*. ses constructeurs et un opérateur de récursion, respectivement dans *Data* à l'aide du produit cartésien et dans *Prop* à l'aide du "et". Il restait alors à poser les axiomes d'induction puis à faire les preuves sur les constructeurs permettant la simplification des termes du C.C. où ils apparaissent avec l'opérateur de récursion.

Le passage d'une version à l'autre ne s'est pas fait aussi facilement qu'on pouvait

l'espérer. Dans la nouvelle version les "axiomes d'induction" associés à un type inductif sont internalisés (avec les réserves que l'on peut faire sur l'emploi de ce mot) et la commande "Elim" les applique de la meilleure façon possible. Les commandes associées à l'égalité permettent des substitutions dont certaines demandaient un trésor d'imagination dans la première version (il y a cependant encore quelques améliorations à apporter). D'autre part, dans les deux versions, la démarche est invariablement constituée de retours en arrière pour démontrer des lemmes nécessaires à la peuve du théorème sur lequel on travaille. La traduction point par point dans la nouvelle version de la preuve faite dans la première a donc comporté dans un premier temps un nombre important de lemmes inutiles, et il a fallu faire le ménage. A ce sujet, on peut dire que la nouvelle version n'oblige pas à se poser autant de questions pour savoir comment on va écrire les choses de façon à les rendre effectivement utilisables, ni à redire de façon différente plusieurs fois la même chose. Ceci dit, il est important d'ajouter que la preuve, même si elle est beaucoup plus courte, a gardé la même structure.

On peut se demander s'il n'est pas plus intéressant de faire certaines preuves dans un cadre plus général que celui du problème que l'on traite, c'est à dire d'utiliser systèmatiquement le polymorphisme permis par le C.C. Cette remarque concerne en particulier les résultats obtenus sur les listes de variables.
Dans un premier temps, on travaille dans une section où on déclare certains types comme variables, où on fait des hypothèses et où on pose des axiomes. Quand on sort de la section, à ces types, hypothèses et axiomes correspondent des variables à instancier. En d'autres termes, des types non dépendants dans la section sont des types dépendants en dehors de celle-ci. Dans un second temps, on utilise ces types dépendants en instanciant les premières variables avec des types inductifs et éventuellement des théorèmes. L'inconvénient de cette façon de faire est que l'on a alors à manipuler des termes très longs (et partiellement instanciés toujours de la même manière). Pour ne pas avoir de problèmes de lisibilité, on peut renommer ces termes, c'est à dire renommer après les avoir partiellement instanciés, toutes les définitions et les théorèmes de la section dépendant des types déclarés variables, des hypothèses et des axiomes. Cela augmente la longueur du programme, et en diminue la clarté. C'est pourquoi, on a préféré ici, travailler directement sur les listes de variables. Les facilités qu'offre n'importe quel éditeur de texte permettent de passer rapidement au cas général. Il faut moins d'un quart d'heure pour écrire une traduction polymorphique des résultats obtenus sur les listes de variables dans la preuve considérée ici.

La structure de la preuve doit être rigoureuse. La première difficulté est bien entendu le choix des structures de données. Dans l'absolu, on peut dire qu'il faut construire des types inductifs les plus simples possibles et définir les prédicats sur ces types faisant apparaître comme des sous-ensembles de ces types les objets que l'on veut représenter. Il faut se donner un maximum d'outils pour manipuler les égalités et autres prédicats, ne pas hésiter à avoir une version inductive et une version récursive des mêmes prédicats. Il faut également, dès qu'une notion est représentée,

résoudre les problèmes de décidabilité qu'on peut lui associer. On passe alors à la preuve proprement dite, après en avoir déterminé les grandes lignes. Il se peut alors que des points délicats se résolvent très facilement grâce à la facilité avec laquelle on peut mener en C.C. les raisonnements inductifs. A l'opposé, des notions simples en dehors du C.C., mais éloignées des constructeurs des types de base, peuvent poser de grandes difficultés et forcer l'utilisateur à des détours parfois inattendus (voir les prédicats *Remove* et *Diffelnb*). Enfin, dans certains cas, la preuve étant faite dans les types de base, on doit prouver lors d'un "second passage" la conservation du résultat dans les sous-ensembles qui représentent réellement les objets sur lesquels doit porter le résultat.

Il reste à parler de la difficulté à communiquer les preuves ou à les reprendre après un certain laps de temps. Les preuves doivent être "lisibles", même par des personnes ayant peu d'expérience du logiciel mais connaissant le rôle des commandes. On doit pouvoir en tirer facilement une preuve mathématique. Le choix des identificateurs et la qualité des commentaires sont comme toujours en informatique fondamentaux. Il est même préférable dans certains cas de faire passer une certaine sémantique avant la rigueur. Si des identificateurs ne sont pas assez explicites ou ont été définis beaucoup plus haut, il est bon de rappeler leur signification au moment de leur utilisation. Contrairement à ce qui se passe en général dans l'écriture des programmes l'indentation présente peu d'avantages sauf dans les définitions. L'indentation ne ferait que mettre en évidence la complexité de certains raisonnements par induction, alors que le système offre justement l'avantage de guider pas à pas l'utilisateur qui n'a pas à se demander où il en est dans sa preuve. Il est préférable d'insérer, une fois la preuve faite, des commentaires.

Références

[1] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin-Mohring, and B. Werner. The Coq Proof Assistant. User's guide, INRIA-CNRS-ENS, 1991.

[2] Z. Manna and R. J. Waldinger. Deductive synthesis of the unification algorithm. *Science of Computer Programming*, 1(1):5–48, 1981.

[3] C. Paulin-Mohring. *Extraction de programmes dans le calcul des constructions*. PhD thesis, Université de Paris VII, 1989.

[4] L. Paulson. Verifying the unification algorithm in lcf. *Science of Computer Programming*, 5:143–169, 1985.

```
(*************************************************************)
(****** Projet Formel - Calculus of Inductives Constructions V5.6 *******)
(*************************************************************)
(***************** uses Prelude.v, Specif.v and Nat.v ********************)
(*************************************************************)
(******************** nat_complements.v ******************************)
(*************** New theorems and lemmas on naturals. ****************)
(*************************************************************)

(*************************************************************)                10
(************* Lemmas on naturals proved in Nat.v and *****************)
(******************** used with the command "auto". ******************)
(*************************************************************)
(*O_S       :(n:nat)(~<nat>O=(S n))*)
(*eq_S      :(n:nat)(m:nat)(<nat>n=m)->(<nat>(S n)=(S m))*)
(*eq_add_S :(n:nat)(m:nat)(<nat>(S n)=(S m))->(<nat>n=m)*)
(*le_pred_n:(n:nat)(le (pred n) n)*)
(*le_n_Sn  :(n:nat)(le n (S n))*)
(*le_S_n   :(n:nat)(m:nat)(le (S n) (S m))->(le n m)*)
(*le_Sn_O  :(n:nat)(~(le (S n) O))*)                                          20
(*le_n     :(n:nat)(le n n)*)
(*le_n_S   :(n:nat)(m:nat)(le n m)->(le (S n) (S m))*)
(*************************************************************)

(*************************************************************)
(********** Logic tools : To translate P:Prop into Q:Set : ***************)
(*************************************************************)

Definition P_S:(A:Set)(P:A->Prop)A->Set=
[A:Set][P:A->Prop][x:A]{a:A|(<A>a=x)&(P a)}.                                  30

Goal (A:Set)(a:A)(P:A->Prop)(P a)->(P_S A P a).
Intros;Unfold P_S;Exists a;Auto.
Save P_S_proof1.

Goal (A:Set)(a:A)(P:A->Prop)(P_S A P a)->(P a).
Unfold P_S;Intros A a P h;Elim h;Intros x h0;Elim h0;Auto.
Save P_S_proof2.

(*************************************************************)                40
(***************** Logic tools  To negate an equality : ******************)
(*************************************************************)

Goal (A:Set)(f:A->Prop)(a,b:A)(f a)->(~(f b))->~<A>a=b.
Unfold not;Intros A f a b H H0 H1;Elim H0;Elim H1;Auto.
Save Diff.

(*** Replace "Apply h;Auto" by "auto" when the Type of h is False : ***)

Goal ~False.                                                                  50
Unfold not;Auto.
Save n_False.
Hint n_False.
(*************************************************************)

(*************************************************************)
(*********************) Section complement_nat. (*********************)
(*************************************************************)

(*************************************************************)                60
```

```
(***************** Decidability of the equality in the Set nat : *************)
(************************************************************************)

Goal (x,y:nat){<nat>x=y}+{~<nat>x=y}.
Induction x.
(*case x=O*)
Induction y;Auto.
(*... case y=O*)(*Apply refl_equal*)
(*... case y=(S z)*)(*Apply O_S*)
(*case x=(S y)*)
Induction y0.
(*... case y0=O*)
Right;Unfold not;Intros;
Absurd <nat>O=(S y);Auto.(*Apply O_S and sym_equal*)
(*... case y0=(S y1)*)
Intros y1 h;Elim (H y1);Intros.
(*... ... case y=y1*)
Auto.(*Apply eq_S*)
(*case not y=y1*)
Right;Unfold not;Intros;Absurd <nat>y=y1;
Auto.(*Apply eq_S*)(*Apply eq_add_S*)
Save nat_eq_decS.

Goal (x,y:nat)(<nat>x=y)\/(~<nat>x=y).
Intros x y;Elim (nat_eq_decS x y);Auto.
Save nat_eq_decP.

(************************************************************************)
(************** General induction (with le) : ***************************)
(************************************************************************)

Goal (n:nat)(P:nat->Set)(P O)->((p:nat)((q:nat)(le q p)->(P q))
->(P (S p)))->(P n).
Intros n P;Cut ((m:nat)(le m n)->(P m))->(P n).
2:Auto.(*Apply le_n*)
Intros h h0 h1;Apply h.
Elim n.
(*case n=O*)
(*... case m=O*)
Induction m.
Intros;Auto.
(*... case m=(S y0)*)
Intros;Absurd (le (S y) O);Auto.(*le_Sn_O*)
(*case n=(S y)*)
Induction m.
(*... case m=O*)
Auto.
(*... case m=(S y0)*)
Intros;Apply (h1 y0).
Intros;Apply H.
Apply le_trans  with y0;Auto.(*le_S_n*)
Save ind_leS.

Goal (n:nat)(P:nat->Prop)(P O)->((p:nat)((q:nat)(le q p)->(P q))
->(P (S p)))->(P n).
Intros;Apply P_S_proof2 with nat.
Apply ind_leS;Intros;Apply P_S_proof1;Auto.
Apply H0;Intros;Elim (H1 q);Intros;Auto.
Elim p0;Auto.
Save ind_leP.
```

```
(*************************************************************************)
(********** Reasoning by cases with the natural constructors : ************)
(*************************************************************************)

Goal (m:nat)(<nat>O=m)\/(<nat>m=(S (pred m))).
Intro:Elim m:Auto.
Save pred_or.

Goal (x:nat)(P:nat->Set)(P O)->((n:nat)(P (S n)))->(P x).                    130
Intros;Elim x;Auto.
Save nat_caseS.

(*************************************************************************)
(************** Decidability of le : *************************************)
(*************************************************************************)

Goal (n,p:nat){(le n p)}+{~(le n p)}.
Induction n;Auto.              (*case n=O*)
Induction p;Auto.              (*case n=(S y)*)(*Apply le_Sn_O*)              140
Intros;Elim (H y0);Intros;Auto.                (*Apply le_n_S*)
Right;Unfold not;Intros;Elim b;Auto.           (*Apply le_S_n*)
Save le_decS.

Goal (n,p:nat)(le n p)\/(~(le n p)).
Intros;Elim (le_decS n p);Intros;Auto.
Save le_decP.

Goal (n,p:nat)(le n p)->{(le (S n) p)}+{<nat>n=p}.
Induction n.                                                                 150
Induction p;Auto.
Induction p.
Intros;Absurd (le (S y) O);Auto.
Intros;Elim (H y0);Auto.(*le_n_S*)
Save le_S_eqS.

Goal (n,p:nat)(le n p)->((le (S n) p)\/(<nat>n=p)).
Intros;Elim (le_S_eqS n p);Intros;Auto.
Save le_S_eqP.
                                                                             160
End complement_nat.
```

```
(*************************************************************************)
(******* Projet Formel – Calculus of Inductives Constructions V5.6 ********)
(*************************************************************************)
(***************** uses Prelude.v, Specif.v, Nat.v ************************)
(********************* and nat_complements.v ******************************)
(*************************************************************************)
(*********************** nat_term_eq_quasiterm.v **************************)
(*************************************************************************)

(*-----------------------------------------------------*)
(*------------- Verifications on the terms:begin ------------*)
(*-----------------------------------------------------*)

Section nat_sequence.

Inductive Definition list_nat:Set=
nil_nat:list_nat
|cons_nat:nat->list_nat->list_nat.

Definition At_last:nat->list_nat->list_nat
=[x:nat][l:list_nat]((<nat->list_nat>Match l
   with [x:nat](cons_nat x nil_nat)
        [c:nat][q:list_nat][f:nat->list_nat]
          [x:nat](cons_nat c (f x))) x).

Definition List_queue_nat:list_nat->list_nat
=[l:list_nat](<list_nat>Match l
                with nil_nat
                     [x:nat][tail:list_nat]
                       [tail_file:list_nat](At_last x tail_file)).

Definition Constr_f:list_nat->nat->nat=
[l:list_nat](<nat->nat>Match l with
    [x:nat]O
    [m:nat][tail:list_nat][f_tail:nat->nat]
       [n:nat](<nat>Match n with m [p:nat][q:nat](f_tail p))).

(*************************************************************************)
(********************** Properties of Constr_f ***************************)
(*************************************************************************)

Inductive Definition OK_Constr_f[f:list_nat->nat->nat]:Prop=
OK_Constr_f_init:((x:nat)<nat>O=(f nil_nat x))
->((val:nat)(l:list_nat)<nat>val=(f (cons_nat val l) O))
->((x,val:nat)(l:list_nat)<nat>(f l x)=(f (cons_nat val l) (S x)))
->(OK_Constr_f f).

Goal (OK_Constr_f Constr_f).
Apply OK_Constr_f_init:Simpl:Auto:Induction x:Simpl:Auto.
Save Constr_f_OK.

Inductive Definition
OK_At_last[f:nat->list_nat->list_nat]:Prop=
OK_At_last_init:
((x:nat)<list_nat>(cons_nat x nil_nat)=(f x nil_nat))
->((x,y:nat)<list_nat>(cons_nat y (cons_nat x nil_nat))
     =(f x (f y nil_nat)))
->(OK_At_last f).

Goal (OK_At_last At_last).
```

```
Apply OK_At_last_init;Simpl;Auto.
Save At_last_OK.

Inductive Definition
OK_List_queue_nat[f:list_nat->list_nat]:Prop=
OK_List_queue_nat_init:(<list_nat>nil_nat=(f nil_nat))
->((x:nat)(l:list_nat)<list_nat>(At_last x (f l))=(f (cons_nat x l)))
->(OK_List_queue_nat f).

Goal (OK_List_queue_nat List_queue_nat).                                    70
Apply OK_List_queue_nat_init;Simpl;Auto.
Save List_queue_nat_OK.

End nat_sequence.

(************************************************************************)
(******************* Specifications of terms ****************************)
(************************************************************************)

Section terms.                                                              80

Definition fun:Set=nat.

Definition var:Set=nat.

Goal (x,x0:var)(<var>x=x0)\/(~<var>x=x0).
Intros;Elim (nat_eq_decP x x0);Auto.
Save var_eq_decP.

Goal (x,x0:var){<var>x=x0}+{~<var>x=x0}.                                    90
Intros;Elim (nat_eq_decS x x0);Auto.
Save var_eq_decS.

Goal (x,x0:fun)(<fun>x=x0)\/(~<fun>x=x0).
Intros;Elim (nat_eq_decP x x0);Auto.
Save fun_eq_decP.

Goal (x,x0:fun){<fun>x=x0}+{~<fun>x=x0}.
Intros;Elim (nat_eq_decS x x0);Auto.
Save fun_eq_decS.                                                           100


Inductive Definition quasiterm:Set=
V:var->quasiterm                                  (*variable quasiterm*)
|C:fun->quasiterm                                 (*constant quasiterm*)
|Root : fun->quasiterm->quasiterm                          (*rooting*)
|ConsArg  : quasiterm->quasiterm->quasiterm.
                                  (*building the pairs of quasiterms*)

(************************************************************************)  110
(************************ The arity function ****************************)
(************************************************************************)

Hypothesis list_arity:list_nat.

Definition arity:fun->nat=(Constr_f (List_queue_nat list_arity)).

(************************************************************************)
(********************** Length of a quasiterm ***************************)
(************************************************************************)  120
```

```
Definition Length :quasiterm->nat=
[t:quasiterm](<nat>Match t with [x:var](S O)
                  [f:fun](S O)
                  [f:fun][u:quasiterm][n:nat](S O)
                  [t1:quasiterm][m:nat][t2:quasiterm][n:nat]
                     (<nat>Match m with n [x:nat]S)).

(*************************************************************************)
(********************** Predicate SIMPLE *****************************)
(****** Simple term is constant term or variable term or rooted term. *******)
(*************************************************************************)

Definition SIMPLE:quasiterm->Prop=
[t:quasiterm](<Prop>Match t with
                  [x:var]True
                  [l:fun]True
                  [l:fun][u:quasiterm][p:Prop]True
                  [t1:quasiterm][p:Prop][t2:quasiterm][q:Prop]False).

(*************************************************************************)
(****************** Predicates l_term and L_TERM : ***********************)
(*************************************************************************)

Inductive Definition l_term:quasiterm->Prop=
l_term_initV:(x:var)(l_term (V x))
|l_term_initC:(f:fun)(<nat>O=(arity f))->(l_term (C f))
|l_term_Root:(f:fun)(t:quasiterm)(l_term t)
->(<nat>(arity f)=(Length t))->(l_term (Root f t))
|l_term_ConsArg:(t1,t2:quasiterm)(l_term t1)->(l_term t2)
->(SIMPLE t1)->(l_term (ConsArg t1 t2)).

Definition L_TERM : quasiterm->Prop
=[t:quasiterm](<Prop>Match t
with [x:var]True
     [c:fun](<nat>O=(arity c))
     [l:fun][t:quasiterm][p:Prop]p/\(<nat>(arity l)=(Length t))
     [t1:quasiterm][p1:Prop][t2:quasiterm][p2:Prop]
     (SIMPLE t1)/\p1/\p2).

(*************************************************************************)
(********************* L_TERM is equivalent to l_term : ******************)
(*************************************************************************)

Goal (t:quasiterm)(L_TERM t)->(l_term t).
Intro;Elim t;Simpl;Intros.
Apply l_term_initV;Auto.
Apply l_term_initC;Auto.
Apply l_term_Root.
Apply H:Elim H0;Auto.
Elim H0;Auto.
Apply l_term_ConsArg.
Apply H:Elim H1;Intros H2 H3;Elim H3:Intros:Auto.
Apply H0:Elim H1;Intros H2 H3:Elim H3:Intros:Auto.
Elim H1:Auto.
Save L_TERM_l_term.

Goal (t:quasiterm)(l_term t)->(L_TERM t).
Intros;Elim H:Simpl;Auto.
```

```
Save l_term_L_TERM.

(*************************************************************************)
(******************** Predicate term : ***********************************)
(*************************************************************************)

Inductive Definition term[t:quasiterm]:Prop=
term_init:(L_TERM t)->(SIMPLE t)->(term t).

(*************************************************************************)                                    190
(************* Decidability of L_TERM and decidability of term : *********)
(*************************************************************************)


Goal (t:quasiterm){(SIMPLE t)}+{~(SIMPLE t)}.
Intros;Elim t;Simpl;Auto.
Save SIMPLE_decS.

Goal (t:quasiterm){(L_TERM t)}+{~(L_TERM t)}.
Induction t;Simpl;Auto.                                                                                        200
Intros f;Elim (nat_eq_decS O (arity f));Intros;Simpl;Auto.
Intros f y h0;Elim (nat_eq_decS (arity f) (Length y));Intros h.
Elim h;Elim h0;Intros;Auto.
Right;Unfold not;Intros h1;Elim h1;Intros;Elim b;Auto.
Right;Unfold not;Intros h1;Elim h1;Intros;Elim h;Auto.
Intros y h y0 h0;Elim h;Intros.
Elim h0;Intros.
Elim (SIMPLE_decS y);Intros h1.
Auto.
Right;Unfold not;Intros h2;Elim h2;Intros.                                                                     210
Elim h1;Unfold not;Intros;Auto.
Right;Unfold not;Intros h2;Elim h2;Intros.
Elim b;Elim H0;Intros;Auto.
Right;Unfold not;Intros h2;Elim h2;Intros.
Elim b;Elim H0;Intros;Auto.
Save L_TERM_decS.

Goal (t:quasiterm){(term t)}+{~(term t)}.
Intro;Elim (L_TERM_decS t);Intros h1;Elim (SIMPLE_decS t);Intros h2.
Left;Apply term_init;Auto.                                                                                     220
Right;Unfold not;Intros h3;Elim h2;Elim h3;Intros;Auto.
Right;Unfold not;Intros h3;Elim h1;Elim h3;Intros;Auto.
Right;Unfold not;Intros h3:Elim h1:Elim h3;Intros;Auto.
Save term_decS.

(*************************************************************************)
(******************** Properties of Length *******************************)
(*************************************************************************)

Goal (t:quasiterm){x:nat|<nat>(Length t)=(S x)}.                                                                230
Induction t.
Exists O;Simpl;Auto.
Exists O;Simpl;Auto.
Exists O;Simpl;Auto.
Intros.
Elim H;Elim H0:Intros.
Simpl:Replace (Length y) with (S x0);
Replace (Length y0) with (S x).
Exists (<nat>Match x0 with (S x) [x:nat]S);Auto.
Save Length_n_O.                                                                                               240
```

```
Goal (t,t0:quasiterm)~<nat>(S O)=(Length (ConsArg t t0)).
Intros;Elim (Length_n_O t);Elim (Length_n_O t0);Intros;Simpl.
Replace (Length t) with (S x0);
Replace (Length t0) with (S x).
Elim x0;Simpl.
Unfold not;Intros;Absurd <nat>O=(S x);Auto.
Intros;Unfold not;Intros;
Absurd <nat>O=(S (<nat>Match y with (S x) [x:nat]S));Auto.
Save n_SO_Length_ConsArg.

Goal (t:quasiterm)(SIMPLE t)->(<nat>(S O)=(Length t)).
Induction t;Simpl;Intros;Auto.
Absurd False;Auto.
Save SIMPLE_SO.

Goal (t:quasiterm)(L_TERM t)
->(<nat>(S O)=(Length t))->(term t).
Induction t;Intros;Apply term_init;Simpl;Auto.
Absurd <nat>(S O)=(Length (ConsArg y y0));Auto;
Apply n_SO_Length_ConsArg;Auto.
Save Length_SO_term.

Goal (t:quasiterm)
(term t)->((L_TERM t)/\(<nat>(S O)=(Length t))).
Induction t.
Intros;Simpl;Auto.
Intros;Simpl.
Split;Auto.
Elim H;Simpl;Intros h;Elim h;Auto.
Simpl;Intros.
Split;Auto.
Elim H0;Simpl;Intros h;Elim h;Intros;Split;Auto.
Simpl;Intros.
Split.
Elim H1;Simpl;Intros;Auto.
Elim H1;Simpl;Intros.
Absurd False;Auto.
Save term_L_TERM_Length.

End terms.

Section eq_quasiterm.

(*************************************************************************)
(************************ Structural predicates ************************)
(*************************************************************************)

Definition BC:quasiterm->Prop=[t:quasiterm](<Prop>Match t
with [x:var]False
     [x:fun]True [l:fun][p:quasiterm][P:Prop]False
     [t1:quasiterm][P1:Prop][t2:quasiterm][P2:Prop]False).

Definition BV:quasiterm->Prop=[t:quasiterm](<Prop>Match t
with [x:var]True
     [x:fun]False [l:fun][p:quasiterm][P:Prop]False
     [t1:quasiterm][P1:Prop][t2:quasiterm][P2:Prop]False).

Definition BRoot:quasiterm->Prop=[t:quasiterm](<Prop>Match t
with [x:var]False
```

```
      [x:fun]False [l:fun][p:quasiterm][P:Prop]True
      [t1:quasiterm][P1:Prop][t2:quasiterm][P2:Prop]False).

Definition BConsArg:quasiterm->Prop=[t:quasiterm](<Prop>Match t
with [x:var]False
      [x:fun]False [l:fun][p:quasiterm][P:Prop]False
      [t1:quasiterm][P1:Prop][t2:quasiterm][P2:Prop]True).

(*************************************************************************)
(********************** Destroyers (Destructors): ************************)
(*************************************************************************)
Definition Destr1=[t:quasiterm](<quasiterm>Match t
with V
     C
     [l:fun][p,q:quasiterm]p
     [p1,p2,q1,q2:quasiterm]p1).

Definition Destr2=[t:quasiterm](<quasiterm>Match t
with V
     C
     [l:fun][p,q:quasiterm]p
     [p1,p2,q1,q2:quasiterm]q1).

Definition Destrvar=[X:var][t:quasiterm](<var>Match t
with [x:var]x
     [l:fun]X
     [l:fun][p:quasiterm][x:var]X
     [p1:quasiterm][x1:var][p2:quasiterm][x2:var]X).


Definition Destrfun=[F:fun][t:quasiterm](<fun>Match t
with [x:var]F
     [l:fun]l
     [l:fun][p:quasiterm][x:fun]l
     [p1:quasiterm][x1:fun][p2:quasiterm][x2:fun]F).

(*************************************************************************)
(******************** Equal in the Set of quasiterms : *******************)
(*************************************************************************)

Goal (l1,l2:fun)(<quasiterm>(C l1)=(C l2))->(<fun>l1=l2).
Intros;Replace l1 with (Destrfun l1 (C l1));Auto.
Replace l2 with (Destrfun l1 (C l2));Auto.
Elim H:Auto.
Save proj_C.

Goal (x1,x2:var)(<quasiterm>(V x1)=(V x2))->(<var>x1=x2).
Intros;Replace x1 with (Destrvar x1 (V x1));Auto.
Replace x2 with (Destrvar x1 (V x2));Auto.
Elim H:Auto.
Save proj_V.

Goal (t1,t2:quasiterm)(l1,l2:fun)
(<quasiterm>(Root l1 t1)=(Root l2 t2))->(<fun>l1=l2).
Intros;Replace l1 with (Destrfun l1 (Root l1 t1));Auto.
Replace l2 with (Destrfun l1 (Root l2 t2));Auto.
Elim H:Auto.
Save proj_Root1.

Goal (t1,t2:quasiterm)(l1,l2:fun)
```

```
(<quasiterm>(Root l1 t1)=(Root l2 t2))->(<quasiterm>t1=t2).
Intros:Replace t1 with (Destr1 (Root l1 t1));Auto.
Replace t2 with (Destr2 (Root l2 t2));Auto.
Elim H;Auto.
Save proj_Root2.

Goal (t1,t2,t3,t4:quasiterm)
(<quasiterm>(ConsArg t1 t2)=(ConsArg t3 t4))->(<quasiterm>t1=t3).
Intros:Replace t1 with (Destr1 (ConsArg t1 t2));Auto.
Replace t3 with (Destr1 (ConsArg t3 t4));Auto.                          370
Elim H;Auto.
Save proj_ConsArg1.

Goal (t1,t2,t3,t4:quasiterm)
(<quasiterm>(ConsArg t1 t2)=(ConsArg t3 t4))->(<quasiterm>t2=t4).
Intros;Replace t2 with (Destr2 (ConsArg t1 t2));Auto.
Replace t4 with (Destr2 (ConsArg t3 t4));Auto.
Elim H;Auto.
Save proj_ConsArg2.
                                                                        380
(*************************************************************************)
(****************** Not equal in the Set of the quasiterms : *************)
(*************************************************************************)
Goal  (l,l0:fun)(~<fun>l=l0)->(~(<quasiterm>(C l)=(C l0))).
Unfold not;Intros;Elim H;Apply proj_C;Auto.
Save C_diff_C.

Goal  (x,y:var)(~(<var>x=y))->(~(<quasiterm>(V x)=(V y))).
Unfold not;Intros;Elim H;Apply proj_V;Auto.
Save V_diff_V.                                                          390

Goal  (t,t0,t1,t2:quasiterm)((~(<quasiterm>t=t1))\/(~(<quasiterm>t0=t2)))
->(~(<quasiterm>(ConsArg t t0)=(ConsArg t1 t2))).
Unfold not;Intros;Elim H;Intros;Elim H1.
Apply proj_ConsArg1 with t0 t2;Auto.
Apply proj_ConsArg2 with t t1;Auto.
Save ConsArg_diff_ConsArg.

Goal  (l,l0:fun)(t,t0:quasiterm)((~(<fun>l=l0))\/(~(<quasiterm>t=t0)))
->(~(<quasiterm>(Root l t)=(Root l0 t0))).                              400
Unfold not;Intros;Elim H;Intros;Elim H1.
Apply proj_Root1 with t t0;Auto.
Apply proj_Root2 with l l0;Auto.
Save Root_diff_Root.

(*************************************************************************)
(************ Decidability of the equality in the Set of quasiterms : *********)
(*************************************************************************)
Goal  (t,t0:quasiterm){<quasiterm>t=t0}+{~(<quasiterm>t=t0)}.
Induction t.                                                            410
Induction t0.
(*t=(V ...)*)
Intros :Elim (var_eq_decS v v0):Intros H.
Elim H:Auto.
Right:Apply V_diff_V:Auto.
Intros:Right:Apply (Diff quasiterm BV):Simpl:Auto.
Intros:Right:Apply (Diff quasiterm BV):Simpl:Auto.
Intros:Right:Apply (Diff quasiterm BV):Simpl:Auto.
(*t=(C ...)*)
Induction t0.                                                           420
```

```
Intros;Right;Apply (Diff quasiterm BC);Simpl;Auto.
Intros;Elim (fun_eq_decS f f0);Intros H.
Elim H;Auto.
Right;Apply C_diff_C;Auto.
Intros;Right;Apply (Diff quasiterm BC);Simpl;Auto.
Intros;Right;Apply (Diff quasiterm BC);Simpl;Auto.
(*t=(Root...)*)
Induction t0.
Intros;Right;Apply (Diff quasiterm BRoot);Simpl;Auto.
Intros;Right;Apply (Diff quasiterm BRoot);Simpl;Auto.                    430
Intros.
Elim (H y0);Intros.
Elim a;Elim (fun_eq_decS f f0);Intros.
Elim a0;Auto.
Right;Apply Root_diff_Root;Auto.
Right;Apply Root_diff_Root;Auto.
Intros;Right;Apply (Diff quasiterm BRoot);Simpl;Auto.
(*t=(ConsArg ...)*)
Induction t0.
Intros;Right;Apply (Diff quasiterm BConsArg);Simpl;Auto.                 440
Intros;Right;Apply (Diff quasiterm BConsArg);Simpl;Auto.
Intros;Right;Apply (Diff quasiterm BConsArg);Simpl;Auto.
Intros.
Elim (H y1);Intros.
Elim (H0 y2);Intros.
Elim a;Elim a0;Auto.
Right;Apply ConsArg_diff_ConsArg;Auto.
Right;Apply ConsArg_diff_ConsArg;Auto.
Save quasiterm_eq_decS.

                                                                         450
Goal  (t,t0:quasiterm)(<quasiterm>t=t0)\/(~<quasiterm>t=t0).
Intros;Elim (quasiterm_eq_decS t t0);Intros;Auto.
Save quasiterm_eq_decP.

(*************************************************************************)
(********************* End of equality in quasiterm **********************)
(*************************************************************************)


(*************************************************************************)   460
(************************) End eq_quasiterm. (*************************)
(*************************************************************************)

(*-----------------------------------------------------*)
(*------------- Verifications on the terms:end ---------------*)
(*-----------------------------------------------------*)
```

```
(*************************************************************************)
(******* Projet Formel – Calculus of Inductives Constructions V5.6 ********)
(*************************************************************************)
(***************** uses Prelude.v, Specif.v, Nat.v, **********************)
(************ nat_complements.v and nat_term_eq_quasiterm.v ***************)
(*************************************************************************)
(********************** is_in_quasiterm_term_subst.v *********************)
(*************************************************************************)

(*************************************************************************)
(******************** VARIABLE IN A QUASITERM ****************************)
(*************************************************************************)

Inductive Definition is_in [x:var]:quasiterm->Prop
=is_in_init : (is_in x (V x))
|is_in_Root : (l:fun)(t:quasiterm)(is_in x t)->(is_in x (Root l t))
|is_in_ConsArg1 : (t1,t2:quasiterm)(is_in x t1)->(is_in x (ConsArg t1 t2))
|is_in_ConsArg2 : (t1,t2:quasiterm)(is_in x t2)
                  ->(is_in x (ConsArg t1 t2)).

Definition IS_IN : var->quasiterm->Prop
= [x:var][t:quasiterm]
(<Prop>Match t with
        [y:var](<var>x=y)
        [l:fun]False
        [l:fun][t:quasiterm][p:Prop]p
        [t1:quasiterm][p1:Prop][t2:quasiterm][p2:Prop](p1\/p2)).

Goal (x:var)(t:quasiterm)(IS_IN x t)->(is_in x t).
Induction t.
Intros.
Elim H;Apply is_in_init.
Intros;Elim H.
Intros;Apply is_in_Root;Auto.
Intros;Elim H1;Intros.
Apply is_in_ConsArg1;Auto.
Apply is_in_ConsArg2;Auto.
Save IS_IN_is_in.

Goal (x:var)(t:quasiterm)(is_in x t)->(IS_IN x t).
Intros;Elim H;Simpl;Auto.
Save is_in_IS_IN.

(*************************************************************************)
(************************ IS_IN decidability *****************************)
(*************************************************************************)

Goal (x:var)(t:quasiterm){(IS_IN x t)}+{~(IS_IN x t)}.
Induction t.
Intro;Elim (var_eq_decS x v);Intros;Simpl;Auto.
Intros;Right;Auto.
Intros f y h;Elim h;Intros;Simpl;Auto.
Intros y h y0 h0;Elim h;Elim h0;Intros;Simpl;Auto.
Right;Unfold not;Intros h1;Elim h1;Intros h2.
Elim b0;Auto.
Elim b;Auto.
Save IS_IN_decS.

Goal (x:var)(t:quasiterm)(IS_IN x t)\/(~(IS_IN x t)).
Intros;Elim (IS_IN_decS x t);Intros;Auto
```

```
Save IS_IN_decP.

(*************************************************************************)
(************************** End of IS_IN *********************************)
(*************************************************************************)

(*************************************************************************)
(********************** QUASISUBSTITUTIONS *******************************)
(*************************************************************************)

Definition quasisubst=var->quasiterm.

Definition Subst:quasisubst->quasiterm->quasiterm
=[f:quasisubst][t:quasiterm](<quasiterm>Match t
with f
     C
     [l:fun][t:quasiterm][u:quasiterm](Root l u)
     [t1,u1,t2,u2:quasiterm](ConsArg u1 u2)).

(*-----------------------------------------------------*)
(*------------- Verifications on the terms:begin ------------*)
(*-----------------------------------------------------*)

Inductive Definition term_subst[l:list_nat;f:quasisubst]:Prop=
term_subst_init:((x:var)(term l (f x)))->(term_subst l f).

(*************************************************************************)
(********************** If (term_subst l f), *****************************)
(********* then Length, SIMPLE, (L_TERM l) and (term l) ******************)
(******************** are stable by (Subst f). ***************************)
(*************************************************************************)

Goal (l:list_nat)(f:quasisubst)(t:quasiterm)
(term_subst l f)-><nat>(Length t)=(Length (Subst f t)).
Induction t;Intros;Elim H;Simpl;Auto.
Intros h;Elim (h v).
Elim (f v);Simpl;Auto.
Intros.
Absurd False;Auto.
Intros.
Elim H;Elim H0;Auto.
Save term_subst_eq_Length.

Goal (l:list_nat)(f:quasisubst)(term_subst l f)
->(t:quasiterm)(SIMPLE t)->(SIMPLE (Subst f t)).
Induction t;Simpl;Auto.
Intros;Elim H;Intros h;Elim (h v);Intros;Auto.
Save SIMPLE_term_subst.

Goal (l:list_nat)(t:quasiterm)(L_TERM l t)
->(f:quasisubst)(term_subst l f)
->(L_TERM l (Subst f t)).
Intros l t H;Cut (l_term l t).
2:Try Apply L_TERM_l_term;Auto.
Intros h;Elim h;Simpl;Auto.
Intros x f h0;Elim h0.
Intros h1;Elim (h1 x);Auto.
Intros.
Split;Auto.
Replace (arity l f) with (Length t0)
```

```
Apply  term_subst_eq_Length with l;Auto.
Intros.
Split;Auto.
Apply SIMPLE_term_subst with l;Auto.
Save L_TERM_term_subst.

Goal (l:list_nat)(t:quasiterm)(term l t)
->(f:quasisubst)(term_subst l f)
->(term l (Subst f t)).
Intros;Apply term_init.                                                    130
Elim H;Intros;Apply L_TERM_term_subst;Auto.
Elim H;Intros;Apply SIMPLE_term_subst with l;Auto.
Save term_term_subst.

(*-----------------------------------------------------*)
(*------------  Verifications on the terms:end ---------------*)
(*-----------------------------------------------------*)

(*************************************************************************)
(************** Properties linked to IS_IN and Subst **********************)   140
(*************************************************************************)

Goal (f:quasisubst)(t:quasiterm)(x:var)((y:var)~(IS_IN x (f y))))
->~(IS_IN x (Subst f t)).
Induction t;Simpl;Auto.
Intros;Unfold not;Intros h;Elim h;Intros.
Elim (H x);Auto.
Elim (H0 x);Auto.
Save n_IS_IN_Subst.
                                                                           150
Goal (f:quasisubst)(t:quasiterm)(x,x0:var)(IS_IN x (f x0))
->(IS_IN x0 t)->(IS_IN x (Subst f t)).
Induction t;Simpl;Auto.
Intros.
Elim H0;Auto.
Intros.
Elim H2;Intros.
Left;Apply (H x x0);Auto.
Right;Apply (H0 x x0);Auto.
Save IS_IN_Subst_IS_IN.                                                    160

(*************************************************************************)
(**** If two quasisubstitutions have the same restriction ****************)
(************** on the set of the variables of t. *************************)
(************** then they transform t in the same quasiterm. **************)
(*************************************************************************)

Goal (t:quasiterm)(f,g:quasisubst)
((x:var)(IS_IN x t)->><quasiterm>(f x)=(g x))
->><quasiterm>(Subst f t)=(Subst g t).                                     170
Induction t;Simpl;Auto.
Intros.
Elim (H f0 g);Auto.
Intros.
Elim (H f g);Elim (H0 f g);Auto.
Save eq_restriction_s_t.

(*************************************************************************)
(******* If two quasisubstitutions transform t in the same quasiterm. ********)
(***** then they have the same restriction on the set of the variables of t.****)   180
```

```
(*************************************************************************)

Goal (t:quasiterm)(f,g:quasisubst)
(<quasiterm>(Subst f t)=(Subst g t))
->((x:var)(IS_IN x t)-><quasiterm>(f x)=(g x)).
Induction t;Simpl;Intros.
Replace x with v;Auto.
Absurd False;Auto.
Apply H;Auto.
Apply proj_Root2 with f f;Auto.                                          190
Elim H2;Intros.
Apply H;Auto.
Apply proj_ConsArg1 with (Subst f y0) (Subst g y0);Auto.
Apply H0;Auto.
Apply proj_ConsArg2 with (Subst f y) (Subst g y);Auto.
Save eq_restriction_t_s.

(*************************************************************************)
(********************** End of quasisubstitutions ************************)
(*************************************************************************)                                                                  200
```

```
(*************************************************************************)
(******* Projet Formel – Calculus of Inductives Constructions V5.6 ********)
(*************************************************************************)
(***************** uses Prelude.v, Specif.v, Nat.v, ***********************)
(************* nat_complements.v, nat_term_eq_quasiterm.v ******************)
(****************** and is_in_quasiterm_term_subst.v ***********************)
(*************************************************************************)
(************************** listv_is_in_lv.v *****************************)
(*************************************************************************)

(*************************************************************************)
(********************** Lists of variables : *****************************)
(*************************************************************************)

Inductive Set listv = Nilv : listv | Consv : var -> listv -> listv.

Definition BNilv:listv->Prop=[l:listv](<Prop>Match l
with True
     [x:var][l:listv][b:Prop]False).

Definition BConsv:listv->Prop=[l:listv](<Prop>Match l
with False
     [x:var][l:listv][b:Prop]True).

Definition Appv:listv->listv->listv=[l0,l1:listv](<listv>Match l0
with l1
     [x:var][l2:listv](Consv x)).

Definition Headv:var->listv->var=[x:var][l:listv](<var>Match l
with x
     [x:var][l1:listv][y:var]x).

Definition Tailv:listv->listv=[l:listv](<listv>Match l
with Nilv
     [x:var][l1:listv][l2:listv]l1).

(*************************************************************************)
(**************** Predicate "to be member of a list" : *******************)
(*************************************************************************)

Inductive Definition is_in_lv[x:var]:listv->Prop=
is_in_lv_init:(l:listv)(is_in_lv x (Consv x l))
|is_in_lv_Consv:(l:listv)(x0:var)(is_in_lv x l)
->(is_in_lv x (Consv x0 l)).

Definition IS_IN_LV:var->listv->Prop
=[x:var][l:listv](<Prop>Match l
with False
     [y:var][l1:listv][p:Prop]p\/<var>x=y).

Goal (l:listv)(x:var)(is_in_lv x l)->(IS_IN_LV x l).
Intros:Elim H:Simpl:Auto.
Save is_in_lv_IS_IN_LV.

Goal (l:listv)(x:var)(IS_IN_LV x l)->(is_in_lv x l).
Induction l:Simpl:Intros.
Elim H.
Elim H0:Intros.
Apply is_in_lv_Consv:Auto.
Elim H1:Apply is_in_lv_init:Auto.
```

```
Save IS_IN_LV_is_in_lv.

(*************************************************************************)
(************ Properties 1) if x belongs to l and is not the head, ***********)
(************************** then x belongs to the tail. *********************)
(*************************************************************************)

Goal (x,x0:var)(l:listv)(IS_IN_LV x (Consv x0 l))->(~<var>x=x0)
->(IS_IN_LV x l).
Induction l;Simpl;Intros.                                                    70
Elim H;Intros;Auto;Elim H0;Auto.
Elim H0:Intros:Auto.
Save IS_IN_LV_n_eq.

(*************************************************************************)
(********************** 2) [x0] contains only x0 . ***********************)
(*************************************************************************)

Goal (x,x0:var)(IS_IN_LV x (Consv x0 Nilv))-><var>x=x0.
Intros:Elim H:Intros h;Elim h;Auto.                                          80
Save IS_IN_LV_Consv_Nilv_eq.

(*************************************************************************)
(********************** 3) Decidability of IS_IN_LV : ********************)
(*************************************************************************)

Goal (x:var)(l:listv){(IS_IN_LV x l)}+{~(IS_IN_LV x l)}.
Induction l:Simpl:Intros.
(*l=Nilv*)
Auto.                                                                        90
(*l=(Consv v y)*)
Elim H:Intros.
Auto.
Elim (var_eq_decS x v);Intros.
(* ... <var>x=v *)
Auto.
(*... ~<var>x=v*)
Right;Unfold not;Intros h;Elim h;Intros.
Elim b;Auto.
Elim b0;Auto.                                                                100
Save IS_IN_LV_decS.

Goal (x:var)(l:listv)(IS_IN_LV x l)\/~(IS_IN_LV x l).
Intros:Elim (IS_IN_LV_decS x l);Intros;Auto.
Save IS_IN_LV_decP.

(*************************************************************************)
(********************** IS_IN_LV and Appv ********************************)
(*************************************************************************)
                                                                             110
Goal (l,l0:listv)(x:var)(IS_IN_LV x l)->(IS_IN_LV x (Appv l l0)).
Induction l:Simpl:Intros.
Elim H:Auto.
Elim H0:Intros:Auto.
Save IS_IN_LV_Appv1.

Goal (l,l0:listv)(x:var)(IS_IN_LV x l0)->(IS_IN_LV x (Appv l l0))
Intros l:Elim l:Simpl:Auto.
Save IS_IN_LV_Appv2.
                                                                             120
```

```
Goal (l,l0:listv)(x:var)(IS_IN_LV x (Appv l l0))
     ->((IS_IN_LV x l)\/(IS_IN_LV x l0)).
Induction l;Simpl;Intros;Auto.
Elim H0;Intros;Auto.
Elim (H l0 x);Auto.
Save IS_IN_LV_Appv_IS_IN_LV.

(**************************************************************************)
```

```
(*************************************************************************)
(******* Projet Formel – Calculus of Inductives Constructions V5.6 ********)
(*************************************************************************)
(****************** uses Prelude.v, Specif.v, Nat.v, ***********************)
(************** nat_complements.v, nat_term_eq_quasiterm.v ******************)
(********* is_in_quasiterm_term_subst.v and listv_is_in_lv.v ****************)
(*************************************************************************)
(************************** term_unif.v **********************************)
(*************************************************************************)

(*************************************************************************)
(** Predicates that count the different elements of a list: ***************)
(**** diffelnb is an inductive definition  ******************************)
(**** The definition of DIFFELNB is recursive. ***************************)
(*************************************************************************)

Inductive Definition diffelnb:listv->nat->Prop=
diffelnb_init:(diffelnb Nilv O)
|diffelnb_Consv_in:(l:listv)(n:nat)(x:var)
     (diffelnb l n)->(IS_IN_LV x l)->(diffelnb (Consv x l) n)
|diffelnb_Consv_n_in:(l:listv)(n:nat)(x:var)
     (diffelnb l n)->(~(IS_IN_LV x l))->(diffelnb (Consv x l) (S n)).

Definition DIFFELNB:listv->nat->Prop=[l:listv](<nat->Prop>Match l
with [x:nat]<nat>O=x
     [y:var][l1:listv][f:nat->Prop]
        [n:nat](<Prop>Match n
         with False
          [p:nat][q:Prop]
            ((IS_IN_LV y l1)/\(f (S p)))\/((~(IS_IN_LV y l1))/\(f p)))).

(*************************************************************************)
(**************** DIFFELNB is equivalent to diffelnb *********************)
(*************************************************************************)

Goal (l:listv)(n:nat)(diffelnb l n)->(DIFFELNB l n).
Intros;Elim H;Simpl;Auto.
Induction n0;Simpl;Auto.
Simpl;Auto.
Elim l0;Simpl;Auto.
Save diffelnb_DIFFELNB.

Goal (l:listv)(n:nat)(DIFFELNB l n)->(diffelnb l n).
Induction l.
Induction n;Simpl;Intros.
Exact diffelnb_init.
Absurd <nat>O=(S y);Auto.                    (*Apply O_S*)
Induction n;Intros.
Elim H0.                          (*(DIFFELNB (Consv v y) O):False*)
Elim H1;Intros h:Elim h;Intros.
Apply diffelnb_Consv_in:Auto.
Apply diffelnb_Consv_n_in:Auto.
Save DIFFELNB_diffelnb.

(*************************************************************************)
(***************** Existence of an integer that counts *******************)
(*************************************************************************)

Goal (l:listv){count:nat|(DIFFELNB l count)}.
Induction l.
```

```
Exists O;Simpl;Auto.  (*case l=Nilv*)
Intros.               (*case l=(Consv v y)*)
Elim H;Intros.
Elim (IS_IN_LV_decS v y);Intros.
Exists x.             (*case v in y:(IS_IN_LV v y)*)
Cut (DIFFELNB y x);Auto.
Elim x;Simpl;Auto.    (*case x=(S p)*)
Cut (IS_IN_LV v y);Auto. (*case x=O*)
Elim y;Simpl;Auto.    (*case v not in y:~(IS_IN_LV v y)*)
Exists (S x);Simpl;Auto.                                          70
Save DIFFELNBor.

(*************************************************************************)
(******** The empty list is the only list that contains 0 element. ***********)
(*************************************************************************)

Goal (l:listv)(DIFFELNB l O)-><listv>Nilv=l.
Induction l;Simpl;Auto.
Intros;Absurd False;Auto.
Save DIFFELNB_O.                                                  80

Goal (l:listv)(n:nat)(x:var)(DIFFELNB (Consv x l) n)->(~<nat>O=n).
Induction n.
Simpl;Intros;Absurd False;Auto.
Intros;Unfold not;Intros h;Absurd <nat>O=(S y);Auto.(*Apply O_S*)
Save DIFFELNB_Consv_n_O.

(*************************************************************************)
(********* Inclusion and strict inclusion for the lists as sets. ***********)
(*************************************************************************)                                                  90

Inductive Definition inclv[l1,l2:listv]:Prop=
inclv_init:((y:var)(IS_IN_LV y l1)->(IS_IN_LV y l2))->(inclv l1 l2).

Inductive Definition st_inclv[l1,l2:listv]:Prop=
st_inclv_init:(x:var)(~(IS_IN_LV x l1))->(IS_IN_LV x l2)
->((y:var)(IS_IN_LV y l1)->(IS_IN_LV y l2))->(st_inclv l1 l2).

Goal (l:listv)~(st_inclv l Nilv).
Intros l;Unfold not;Intros h;Elim h;Auto.                         100
Save n_st_inclv_l_Nilv.

(*************************************************************************)
(************************** Remove : ******************************)
(**************** remove is inductively defined, ***********************)
(**************** REMOVE is recursively defined. ***********************)
(*************************************************************************)

Inductive Definition remove[x:var]:listv->listv->Prop=
remove_init:(remove x Nilv Nilv)                                  110
|remove_eq:(l,l0:listv)(remove x l l0)->(remove x (Consv x l) l0)
|remove_n_eq:(l,l0:listv)(x0:var)(remove x l l0)->(~<var>x=x0)
->(remove x (Consv x0 l) (Consv x0 l0)).

Definition REMOVE:var->listv->listv->Prop=
[x:var][l:listv](<listv->Prop>Match l
with [l1:listv]<listv>Nilv=l1
     [v:var][l1:listv][f:listv->Prop]
        [l2:listv]((<var>x=v)/\(f l2))
          \/((~<var>x=v)/\(<Prop>Match l2                         120
```

```
        with False
        [w:var][l3:listv][p:Prop](<var>v=w)/\(f l3)))).

(*************************************************************************)
(************** REMOVE is equivalent to remove ***********************)
(*************************************************************************)

Goal (l,l0:listv)(x:var)(remove x l l0)->(REMOVE x l l0).
Intros;Elim H;Intros;Simpl;Auto.                                          130
Save remove_REMOVE.

Goal (l,l0:listv)(x:var)(REMOVE x l l0)->(remove x l l0).
Induction l.
Intros;Elim H;Apply remove_init.
Induction l0.
Simpl;Intros.
Elim H0;Intros h1;Elim h1;Intros.
Elim H1;Apply remove_eq;Auto.
Absurd False;Auto.                                                        140
Intros.
Elim H1;Intros.
Elim H2;Intros h h0;Elim h;
Apply remove_eq;Auto.
Elim H2;Intros h h0;Elim h0;Intros h1 h2;Elim h1;
Apply remove_n_eq;Auto.
Save REMOVE_remove.

(*************************************************************************)
(******************** REMOVE and not IS_IN_LV ****************************)                150
(*************************************************************************)

Goal (l,l0:listv)(x:var)(REMOVE x l l0)->(~(IS_IN_LV x l))
->~<listv>l=l0.
Intros l l0 x h;Cut (remove x l l0).
2:Apply REMOVE_remove;Auto.
Intros h0;Elim h0;Auto.
Intros;Elim H1;Simpl;Auto.
Simpl;Intros.
Elim H0;Auto;Unfold not;Intros;Elim H2;Auto.                               160
Save REMOVE_n_IS_IN_LV_eq.


(*************************************************************************)
(********************** REMOVE and Consv *********************************)
(*************************************************************************)

Goal (l,l0:listv)(x:var)(REMOVE x (Consv x l) l0)->(REMOVE x l l0).
Intros until x;Simpl;Intros.
Elim H;Intros h;Elim h;Intros;Auto.                                        170
Absurd <var>x=x;Auto.
Save REMOVE_Consv_eq.

Goal (l,l0:listv)(x,x0:var)(REMOVE x (Consv x0 l) (Consv x0 l0))
->(~<var>x0=x)->(REMOVE x l l0).
Intros l l0 x x0;Simpl;Intros;Auto.
Elim H;Intros h;Elim h;Intros.
Absurd <var>x0=x;Auto.
Elim H2;Auto.
Save REMOVE_Consv_n_eq.                                                    180
```

```
(*********************************************************************)
(************** Existence of an l0 such that (REMOVE l l0) ***********)
(*********************************************************************)

Goal (l:listv)(x:var){l0:listv|(REMOVE x l l0)}.
Induction l;Intros.
Exists Nilv;Simpl;Auto.
Elim (H x);Intros l0 h.
Elim (var_eq_decS x v);Intros h0.                                    190
Elim h0;Exists l0;Simpl.
Left;Split;Auto.
Exists (Consv v l0);Simpl.
Right;Split;Auto.
Save sig_REMOVE.

Goal (l:listv)(x:var)<listv>Ex([l0:listv](REMOVE x l l0)).
Intros;Elim (sig_REMOVE l x);Intros l0 h;Exists l0;Auto.
Save exi_REMOVE.
                                                                     200
(*********************************************************************)
(************************ Properties of REMOVE ***********************)
(*********************************************************************)

Goal (l,l0:listv)(x,x0:var)(REMOVE x (Consv x0 l) l0)
     ->(~<var>x=x0)->(~<listv>Nilv=l0).
Induction l0;Intros.
Elim H;Intros h;Elim h;Intros.
Absurd <var>x=x0;Auto.
Absurd False;Auto.                                                   210
Apply Diff with BNilv;Auto;Simpl;Auto.
Save Headv_REMOVE_Consv_Nilv.

Goal (l,l0:listv)(x,x0:var)(REMOVE x (Consv x0 l) l0)->(~<var>x=x0)
     ->((~<listv>Nilv=l0)/\<var>x0=(Headv x l0)).
Intros;Split.
Apply Headv_REMOVE_Consv_Nilv with l x x0;Auto.
Cut (REMOVE x (Consv x0 l) l0).
2:Auto.
Elim l0.                                                             220
Simpl;Intros h;Elim h.
Intros h0:Elim h0;Auto.
Intros h0;Elim h0;Intros;Absurd False;Auto.
Intros;Simpl.
Elim H2;Intros.
Elim H3;Intros.
Absurd <var>x=x0;Auto.
Elim H3;Intros h1 h2;Elim h2;Auto.
Save REMOVE_Consv_n_eq_Headv.
                                                                     230
(*********************************************************************)
(********** Properties linked to REMOVE and IS_IN_LV **************)
(*********************************************************************)

Goal (l,l0:listv)(x,x0:var)(REMOVE x l l0)->(IS_IN_LV x0 l0)->(IS_IN_LV x0 l).
Intros l l0 x x0 h:Cut (remove x l l0).
2:Apply REMOVE_remove;Auto.
Intro:Elim H:Simpl;Auto.
Intros.
Elim H3;Intros;Auto.                                                 240
```

```
Save REMOVE_IS_IN_LV_IS_IN_LV.

Goal (l,l0:listv)(x:var)(REMOVE x l l0)->(x0:var)(~<var>x=x0)
     ->(IS_IN_LV x0 l)->(IS_IN_LV x0 l0).
Intros l l0 x H;Cut (remove x l l0).
2:Apply REMOVE_remove;Auto.
Intro;Elim H0;Simpl;Auto.
Intros.
Elim H4;Auto.
Intros;Absurd <var>x=x0;Auto.                                          250
Intros.
Elim H5;Auto.
Save REMOVE_n_eq_IS_IN_LV_IS_IN_LV.

(*************************************************************************)
(************************* REMOVE and st_inclv : *************************)
(************************ if x does not belong to l **********************)
(******* and x belongs to l0 and l st_inclv l0, and (REMOVE x l0 l1) ******)
(************************** then l st_inclv l1. **************************)
(*************************************************************************)                                                              260

Goal (l,l0,l1:listv)(x:var)(st_inclv (Consv x l) l0)
     ->(REMOVE x l0 l1)->(~(IS_IN_LV x l))->(st_inclv l l1).
Intros l l0 l1 x h;Elim h.
Simpl;Intros.
Apply st_inclv_init with x0.
Unfold not;Intros;Apply H;Auto.
Cut ~<var>x0=x.
Intros;Apply (REMOVE_n_eq_IS_IN_LV_IS_IN_LV l0 l1 x);Auto.
Unfold not;Intros;Elim H4;Auto.                                        270
Unfold not;Intros;Apply H;Auto.
Intros;Cut ~<var>x=y.
Intros;Apply (REMOVE_n_eq_IS_IN_LV_IS_IN_LV l0 l1 x);Auto.
Unfold not;Intros;Apply H3;Replace x with y;Auto.
Save st_inclv_Consv_REMOVE_n_IS_IN_LV_st_inclv.

(*************************************************************************)
(********************** REMOVE and DIFFELNB **********************)
(*************************************************************************)

                                                                       280
Goal (l:listv)(n:nat)(DIFFELNB l n)
     ->(l0:listv)(x:var)(REMOVE x l l0)->(IS_IN_LV x l)->(DIFFELNB l0 (pred n)).
Intros l n h;Cut (diffelnb l n).
2:Apply DIFFELNB_diffelnb:Auto.
Intros H;Elim H.
(* Case1 :  l=Nilv *)
Simpl;Intros;Absurd False;Auto.
(* Case2 l=(Consv x l0) and (IS_IN_LV x l0) *)
Intros until x0;Elim (var_eq_decP x x0):Intro.
(* Case2 ... H3:<var>x=x0*)                                             290
Elim H3;Intros.
Apply H1 with x;Auto.
Apply REMOVE_Consv_eq:Auto.
(* Case2 ... H3:~<var>x=x0*)
Elim l00.
(* Case2 ... H3:~<var>x=x0:l00=Nilv**)
Simpl;Intros.
Elim H4:Intros h0:Elim h0:Intros.
Elim H3;Auto.
Elim H7.                                                               300
```

```
(* Case2 ... H3:~<var>x=x0;l00=(Consv v y))*)
Intros v y h0 h1;Elim h1;Intros.
Elim H4;Intros;Absurd <var>x=x0;Auto.
Elim H4;Intros h2 h3;Elim h3;Intros h4 h5;Elim h4.
Apply diffelnb_DIFFELNB;Apply diffelnb_Consv_in.
Apply DIFFELNB_diffelnb;Apply (H1 y x0);Auto.
Elim H5;Intros;Auto;Absurd <var>x=x0;Auto.
Apply REMOVE_n_eq_IS_IN_LV_IS_IN_LV with l0 x0;Auto.
(* Case3 l=(Consv x l0) and ~(IS_IN_LV x l0) *)
Induction l00.                                                          310
(* Case3 ...l00=Nilv*)·
Intros.
Elim H3;Intros h0;Elim h0;Intros.
Cut ~(IS_IN_LV x l0).
2:Auto.
Elim H5:Intros;Cut <listv>l0=Nilv.
2:Apply REMOVE_n_IS_IN_LV_eq with x0;Auto.
Intros;Cut (DIFFELNB l0 n0).
2:Apply diffelnb_DIFFELNB;Auto;
Replace l0 with Nilv;Simpl;Auto.                                        320
Elim H8;Simpl;Auto.
Elim H6.
(*Case3 ...l00=(Consv v y)*)
Intros.
Change (DIFFELNB (Consv v y) n0).
Elim H4;Intros.
(*Case3 ...l00=(Consv v y))and (<var>x0=x) *)
(*            and (REMOVE x0 l0 (Consv v y))*)
Elim H6;Intros.
Cut (DIFFELNB l0 n0).                                                   330
2:Apply diffelnb_DIFFELNB;Auto.
Cut ~(IS_IN_LV x l0).
2:Auto.
Elim H7;Intros;Replace (Consv v y) with l0;
Try Apply REMOVE_n_IS_IN_LV_eq with x0;Auto.
(*Case3 ...l00=(Consv v y))and ~<var>x0=x*)
(*                          and <var>x=v) and (REMOVE x0 l0 y) *)
Elim H6;Intros h0 h1;Elim h1;Intros.
Elim H5:Intros.
Cut (DIFFELNB y (pred n0)).                                             340
2:Apply (H1 y x0);Auto.
Cut (DIFFELNB l0 n0).
2:Apply diffelnb_DIFFELNB;Auto.
Pattern n0:Apply (nat_case n0).
Simpl;Intros;Absurd (IS_IN_LV x0 l0);Auto.
Replace l0 with Nilv;Try Apply DIFFELNB_O;Auto.
Simpl;Intros.
Elim H7:Right;Split;Auto.
Unfold not;Intros;Elim H2;Apply (REMOVE_IS_IN_LV_IS_IN_LV l0 y x0):Auto.
Absurd <var>x0=x;Auto.                                                  350
Save REMOVE_n_IS_IN_LV_DIFFELNB_pred.

(*************************************************************************)
(************************* st_inclv and DIFFELNB *************************)
(*************************************************************************)

Goal (l:listv)(n:nat)(DIFFELNB l n)->(l0:listv)(n0:nat)(DIFFELNB l0 n0)
     ->(st_inclv l l0)->(le (S n) n0).
Intros l n hyp:Cut (diffelnb l n).
2:Apply DIFFELNB_diffelnb:Auto.                                         360
```

```
Intros h;Elim h.
(* Case1 :  l=Nilv *)
Intros l0 n0;Pattern n0:Apply (nat_case n0).
Simpl;Intros.
Elim H0;Intros;Cut (DIFFELNB l0 O);Auto.
Intros;Absurd (IS_IN_LV x l0);Auto.
Cut <listv>Nilv=l0.
Intros H5;Elim H5;Simpl;Auto.
Apply DIFFELNB_O;Auto.
Intros;Auto.(* Apply le_n_S ; Apply le_O_n *)                                    370
(* Case2 l=(Consv x l0) and (IS_IN_LV x l0) *)
Intros until n00;Pattern n00;Apply (nat_case n00).
Intros h0 h1;Elim h1;Intros.
Absurd (IS_IN_LV x0 l00);Auto.
Cut <listv>Nilv=l00.
Intros H5;Elim H5;Simpl;Auto.
Apply DIFFELNB_O;Auto.
Intros;Apply (H0 l00 (S m));Auto.
Elim H3;Intros.
Apply st_inclv_init with x0;Auto.                                              380
Unfold not;Intros;Elim H4;Simpl;Auto.
Intros;Apply H6;Simpl;Auto.
(* Case3 l=(Consv x l0) and ~(IS_IN_LV x l0) *)
Intros until n00;Pattern n00;Apply (nat_case n00).
Intros h0 h1;Elim h1;Intros.
Absurd (IS_IN_LV x0 l00);Auto.
Cut <listv>Nilv=l00.
Intros H5;Elim H5;Simpl;Auto.
Apply DIFFELNB_O;Auto.
Intros;Apply le_n_S.                                                           390
Elim (exi_REMOVE l00 x);Intros l1 H5.
Apply (H0 l1).
Replace m with (pred (S m));
Try Apply REMOVE_n_IS_IN_LV_DIFFELNB_pred with l00 x;Auto.
Elim H3;Intros.
Apply (H7 x);Simpl;Auto.
Apply st_inclv_Consv_REMOVE_n_IS_IN_LV_st_inclv with l00 x;Auto.
Save DIFFELNB_st_inclv_le_S.

(*************************************************************************)     400
(************************* inclv and DIFFELNB ***************************)
(*************************************************************************)

Goal (l:listv)(n:nat)(DIFFELNB l n)->(l0:listv)(n0:nat)(DIFFELNB l0 n0)
     ->(inclv l l0)->(le n n0).
Intros l n hyp;Cut (diffelnb l n).
2:Apply DIFFELNB_diffelnb;Auto.
Intros h:Elim h.
(* Case1 :  l=Nilv *)
Auto.(*Apply le_O_n*)                                                          410
(* Case2 l=(Consv x l0) and (IS_IN_LV x l0) *)
Intros.
Apply H0 with l00:Auto.
Apply inclv_init:Intros:Elim H3;Simpl:Auto.
(* Case3 l=(Consv x l0) and ~(IS_IN_LV x l0) *)
Intros until n00:Pattern n00:Apply (nat_case n00).
Intros:Absurd (IS_IN_LV x l00).
Cut <listv>Nilv=l00.
2:Apply DIFFELNB_O:Auto.
Intros l14:Elim H4:Simpl:Auto.                                                 420
```

```
Elim H3;Simpl;Auto.
Intros;Elim (exi_REMOVE l00 x);Intros.
Apply le_n_S;Apply H0 with x0.
Replace m with (pred (S m));Auto.
Apply REMOVE_n_IS_IN_LV_DIFFELNB_pred with l00 x;Auto.
Elim H3;Simpl;Auto.
Apply inclv_init;Intros.
Apply REMOVE_n_eq_IS_IN_LV_IS_IN_LV with l00 x;Auto.
Unfold not;Intros h0;Absurd (IS_IN_LV y l0);Auto;Elim h0;Auto.
Elim H3;Simpl;Auto.                                                    430
Save inclv_le.

(*************************************************************)
(********************* (infv t u) is true ********************)
(********* if every variable of t is a variable of u and *********)
(***** if there exists a variable of u which is not a variable of t. ********)
(************ One reads t is less than u by its variables. ***************)
(*************************************************************)

Inductive Definition infv[t1,t2:quasiterm]:Prop=                       440
infv_init:(x:var)(~(IS_IN x t1))->(IS_IN x t2)
     ->((y:var)(IS_IN y t1)->(IS_IN y t2))->(infv t1 t2).

(*************************************************************)
(********** A closed quasiterm has no variable ***********************)
(*************************************************************)

Inductive Definition clos[t:quasiterm]:Prop=
clos_init:((x:var)(~(IS_IN x t)))->(clos t).
                                                                       450
(*************************************************************)
(*********** The list of the variables that occur in a quasiterm *********)
(*************************************************************)

Definition list_var:quasiterm->listv=[t:quasiterm](<listv>Match t
with [x:var](Consv x Nilv)
     [l:fun]Nilv
     [l:fun][t0:quasiterm][l:listv]l
     [t1:quasiterm][l1:listv][t2:quasiterm][l2:listv](Appv l1 l2)).
                                                                       460
(*************************************************************)
(********** (IS_IN x t) and (IS_IN_LV x (list_var t)) are equivalent. *****)
(*************************************************************)

Goal (t:quasiterm)(x:var)(IS_IN x t)->(IS_IN_LV x (list_var t)).
Induction t;Simpl;Intros;Auto.
(*Case t=(V v) : Apply or_is_inror : Assumption
Case t=(C c) : Assumption
Case t=(Root f y) : Apply H : Assumption *)
(*Case t=(ConsArg y y0) *)                                              470
Elim H1;Intros.
Apply IS_IN_LV_Appv1;Auto.
Apply IS_IN_LV_Appv2;Auto.
Save IS_IN_IS_IN_LV.

Goal (t:quasiterm)(x:var)(IS_IN_LV x (list_var t))->(IS_IN x t).
Induction t;Simpl;Intros;Auto.
(*Case t=(C c) : Assumption
Case t=(Root f y) : Apply H : Assumption *)
(*Case t=(V v)*)                                                        480
```

```
Elim H;Intros;Auto;Absurd False;Auto.
(*Exact n_False; Assumption; Assumption *)
(*Case t=(ConsArg y y0) *)
Elim (IS_IN_LV_Appv_IS_IN_LV (list_var y) (list_var y0) x H1);Intros;Auto.
Save IS_IN_LV_IS_IN.

(*************************************************************************)
(********** To count the different variables of a quasiterm  *************)
(*************************************************************************)
Goal (t:quasiterm)(DIFFELNB (list_var t) O)->(clos t).
Intros;Apply clos_init.
Unfold not;Intros x h;Absurd (IS_IN_LV x (list_var t)).
Replace (list_var t) with Nilv;
Try Apply (DIFFELNB_O (list_var t));Auto.
Apply IS_IN_IS_IN_LV;Auto.
Save DIFFELNB_O_clos.

(*************************************************************************)
(********************** clos and ConsArg. ********************************)
(*************************************************************************)

Goal (t1,t2:quasiterm)(clos (ConsArg t1 t2))->(clos t1).
Intros;Elim H;Intros;Apply clos_init.
Intros x;Unfold not;Intros;Elim (H0 x):Simpl;Auto.
Save closConsArg1.

Goal (t1,t2:quasiterm)(clos (ConsArg t1 t2))->(clos t2).
Intros;Elim H;Intros;Apply clos_init.
Intros x;Unfold not;Intros;Elim (H0 x);Simpl;Auto.
Save closConsArg2.

Goal (t1,t2:quasiterm)(clos (ConsArg t1 t2))->((clos t1)/\(clos t2)).
Intros;Elim H;Intros;Split;Apply clos_init.
Intros x;Unfold not;Intros;Elim (H0 x);Simpl;Auto.
Intros x;Unfold not;Intros;Elim (H0 x);Simpl;Auto.
Save closConsArg.

Goal (t:quasiterm)(clos t)->(DIFFELNB (list_var t) O).
Induction t.
(*Case t=(V v) *)
Intros;Elim H.
Simpl;Intros;Absurd <var>v=v;Auto.
(*Case t=(C c) *)
Intros;Simpl;Auto.
(*Case t=(Root f y) *)
Intros;Simpl;Auto.
(*Case t=(ConsArg y y0) *)
Simpl;Intros.
Elim (closConsArg y y0 H1);Intros.
Replace (list_var y) with Nilv;Try Apply DIFFELNB_O;Auto.
Save clos_DIFFELNBO.

Goal (t0,t1:quasiterm)(infv t0 t1)->
(st_inclv (list_var t0) (list_var t1)).
Intros;Elim H;Intros.
Apply st_inclv_init with x.
Unfold not;Intros;Elim H0;Apply IS_IN_LV_IS_IN;Auto.
Apply IS_IN_IS_IN_LV;Auto.
Intros;Apply IS_IN_IS_IN_LV;Apply H2;Apply IS_IN_LV_IS_IN;Auto.
Save infv_st_inclv.
```

```
Goal (t0,t1:quasiterm)(st_inclv (list_var t0) (list_var t1))
     ->(infv t0 t1).
Intros;Elim H;Intros.
Apply infv_init with x.
Unfold not;Intros;Elim H0;Apply IS_IN_IS_IN_LV;Auto.
Apply IS_IN_LV_IS_IN;Auto.
Intros;Apply IS_IN_LV_IS_IN;Apply H2;Apply IS_IN_IS_IN_LV;Auto.
Save st_inclv_infv.

Goal (t:quasiterm)(clos_t)->(t0:quasiterm)~(infv t0 t).
Unfold not:Intros.
Elim H;Intros.
Elim H0:Intros.
Elim (H1 x);Auto.
Save n_infv_t_clos.

(*************************************************************************)
(*************** Quasisubstitution extensively equal to V ****************)
(*************************************************************************)

Goal (f:quasisubst)(t:quasiterm)((x:var)<quasiterm>(V x)=(f x))
     -><quasiterm>t=(Subst f t).
Induction t;Simpl;Auto;Intros.
Elim H;Auto.
Elim H;Elim H0;Auto.
Save eq_V_stab.

Goal (t:quasiterm)<quasiterm>t=(Subst V t).
Intros;Apply eq_V_stab;Auto.
Save V_stab.

Goal (t:quasiterm)(f:quasisubst)(clos t)->(<quasiterm>t=(Subst f t)).
Intros;Elim H;Intros.
Apply trans_equal with (Subst V t);
Try Apply V_stab.
Apply eq_restriction_s_t;Intros;Absurd (IS_IN x t);Auto.
Save clossubst.

(*************************************************************************)
(****************** Predicate dom : (dom f x) is true ********************)
(******************** if f(x) is not equal to V(x). **********************)
(*************************************************************************)

Definition dom:quasisubst->var->Prop=
[f:quasisubst][x:var]~<quasiterm>(V x)=(f x).

Goal (f:quasisubst)(x:var)(~(dom f x))-><quasiterm>(V x)=(f x).
Intros f x:Elim (quasiterm_eq_decP (V x) (f x));Intros;Auto.
Elim H0:Auto.
Save n_dom.

Goal (f:quasisubst)(x:var)(dom f x)\/(~(dom f x)).
Intros f x:Elim (quasiterm_eq_decP (V x) (f x));Intros;Auto.
Right:Unfold not;Intros;Absurd <quasiterm>(V x)=(f x);Auto.
Save dom_decP.

(*************************************************************************)
(****************** Predicate range : (range f x) is true ****************)
(****************** if the variable x is in the image ********************)
```

```
(****************** of an other variable under f. ************************)
(*********************************************************************)

Inductive Definition range[s:quasisubst;x:var]:Prop=
range_init:(y:var)(dom s y)->(IS_IN x (s y))->(range s x).

Goal (f:quasisubst)(x:var)(~(range f x))
     ->(y:var)(dom f y)->~(IS_IN x (f y)).
Unfold not;Intros.
Elim H;Apply range_init with y;Auto.                                    610
Save n_range_n_IS_IN.

Goal (f:quasisubst)(x,y:var)(~(range f x))->(IS_IN x (f y))-><var>x=y.
Intros.
Elim (dom_decP f y);Intros h.
(*Case (dom f y) *)
Absurd (IS_IN x (f y));Auto.
Unfold not;Intros;Elim H;Intros.
Apply range_init with y;Auto.
(*Case ~(dom f y) *)                                                    620
Change (IS_IN x (V y)).
Replace (V y) with (f y);Auto.
Elim (n_dom f y);Auto.
Save n_range_IS_IN_eq.

Goal (f:quasisubst)(t:quasiterm)(x:var)(~(range f x))
     ->(IS_IN x (Subst f t))->(IS_IN x t).
Induction t;Simpl;Intros;Auto.
Simpl;Intros;Apply n_range_IS_IN_eq with f;Auto.
Elim H2;Intros;Auto.                                                   630
Save n_range_IS_IN_Subst.

(*********************************************************************)
(****************** Predicate over : (over f t) is true **************)
(****************** if f does not modify the variables ***************)
(****************** which are not in t. ******************************)
(*********************************************************************)

Definition over : quasisubst->quasiterm->Prop=
[f:quasisubst][t:quasiterm](x:var)(~(IS_IN x t))                       640
-><quasiterm>(V x)=(f x).

(*********************************************************************)
(****************** Predicate under : (under f t) is true ************)
(****************** if every variable which is in the image **********)
(****************** of an other variable under f is in t. ************)
(*********************************************************************)

Definition under : quasisubst->quasiterm->Prop=
[s:quasisubst][t:quasiterm](x,y:var)(dom s y)                          650
->(IS_IN x (s y))->(IS_IN x t).

(*********************************************************************)
(****************** If (under f t) is true then **********************)
(****************** every variable which is in the *******************)
(****************** image of t under the quasisubstitution f *********)
(****************** was before in t. *********************************)
(*********************************************************************)

Goal (f:quasisubst)(t:quasiterm)(x:var)(IS_IN x (Subst f t))           660
```

```
    ->(var)Ex([y:var](IS_IN y t)/\(IS_IN x (f y))).
Induction t;Simpl;Intros;Auto.(*Case t=(Root f u) *)
(*Case t=(V v) *)
Exists v;Auto.
(*Case t=(C c) *)
Elim H.
(*Case t=(ConsArg y y0) *)
Elim H1;Intros.
Elim H with x;Intros;Auto.
Elim H3;Intros;Exists x0;Auto.
Elim H0 with x;Intros;Auto.
Elim H3;Intros;Exists x0;Auto.
Save IS_IN_Subst_exi_IS_IN.

Goal (f:quasisubst)(t:quasiterm)(under f t)
    ->(x:var)(IS_IN x (Subst f t))
    ->(IS_IN x t).
Unfold under;Intros.
Elim (IS_IN_Subst_exi_IS_IN f t x H0);Intros x0 h;Elim h;Intros.
Elim (var_eq_decP x0 x);Intros h0.
Elim h0;Auto.
Apply (H x x0);Auto.
Unfold dom;Unfold not;Intros.
Elim h0;Symmetry;Change (IS_IN x (V x0)).
Replace (V x0) with (f x0);Auto.
Save under_IS_IN_Subst_IS_IN.

(*************************************************************************)
(***************** Strict order sub_term *********************************)
(***************** (usual sense). ****************************************)
(*************************************************************************)

Inductive Definition sub[u:quasiterm]:quasiterm->Prop=
    subConsArgl:(t:quasiterm)(sub u (ConsArg u t))
|subConsArgr:(t:quasiterm)(sub u (ConsArg t u))
|subRoot:(l:fun)(sub u (Root l u))
|subConsArgl2:(t,v:quasiterm)(sub u t)->(sub u (ConsArg t v))
|subConsArgr2:(t,v:quasiterm)(sub u t)->(sub u (ConsArg v t))
|subRoot2:(t:quasiterm)(l:fun)(sub u t)->(sub u (Root l t)).

(*************************************************************************)
(***************** Recursive definition : ********************************)
(*************************************************************************)

Definition SUP:quasiterm->quasiterm->Prop=
[t:quasiterm](<quasiterm->Prop>Match t
with [x:var][u:quasiterm]False
    [l:fun][u:quasiterm]False
    [l:fun][u1:quasiterm][sup1:quasiterm->Prop][u:quasiterm]
        ((<quasiterm>u1=u)\/(sup1 u))
    [u1:quasiterm][sup1:quasiterm->Prop]
        [u2:quasiterm][sup2:quasiterm->Prop]
        [u:quasiterm]((<quasiterm>u1=u)\/(<quasiterm>u2=u)
                \/(sup1 u)\/(sup2 u))).

Definition SUB:quasiterm->quasiterm->Prop=
[t1,t2:quasiterm](SUP t2 t1).

(*************************************************************************)
(***************** SUB is equivalent to sub ******************************)
```

```
(*************************************************************************)

Goal (t1,t2:quasiterm)(sub t1 t2)->(SUB t1 t2).
Intros;Elim H;Unfold SUB;Simpl;Auto.
Save sub_SUB.

Goal (t1,t2:quasiterm)(SUB t1 t2)->(sub t1 t2).
Induction t2;Intros.
(*Case t=(V v) *)
Elim H.
(*Case t=(C c) *)
Elim H.
(*Case t=(Root f y) *)
Elim H0;Intros.
Elim H1;Apply subRoot.
Apply subRoot2;Auto.
(*Case t=(ConsArg y y0) *)
Elim H1;Intros.
Elim H2;Apply subConsArgl.
Elim H2;Intros.
Elim H3;Apply subConsArgr.
Elim H3;Intros.
Apply subConsArgl2;Auto.
Apply subConsArgr2;Auto.
Save SUB_sub.

(*************************************************************************)
(***************** Transitivity of SUP and SUB : ********************)
(*************************************************************************)

Goal (v,u,w:quasiterm)(SUP v u)->(SUP w v)->(SUP w u).
Induction v.
Intros;Absurd False;Auto.(*Case v=(V v0) *)
Intros;Absurd False;Auto.(*Case v=(C f) *)
(*Case v=(Root f y)*)
Induction w.
Intros;Absurd False;Auto.(*Case w=(V v0) *)
Intros;Absurd False;Auto.(*Case w=(C f0) *)
(*Case v=(Root f y);w=(Root f0 y0)*)
Intros.
Elim H2;Intros;Elim H1;Intros;Simpl;Auto.
Elim H4;Replace y0 with (Root f y);Simpl;Auto.
Replace y0 with (Root f y);Simpl;Auto.
(*Case v=(Root f y);w=(ConsArg y0 y1)*)
Intros.
Elim H3;Intros.
Replace y0 with (Root f y);Simpl;Auto.
Elim H4;Intros.
Replace y1 with (Root f y);Simpl;Auto.
Elim H5;Intros;Simpl;Auto.
(*Case v=(ConsArg y y0)*)
Induction w.
Intros;Absurd False;Auto.(*Case w=(V v0) *)
Intros;Absurd False;Auto.(*Case w=(C f0) *)
(*Case v=(ConsArg y y0);w=(Root f0 y0)*)
Intros.
Elim H3;Intros.
Replace y1 with (ConsArg y y0);Simpl;Auto.
Simpl;Auto.
(*Case v=(ConsArg y y0);w=(ConsArg y0 y1)*)
```

```
Intros.
Elim H4;Intros.
Elim H3;Intros.
Elim H6;Intros;Replace y1 with (ConsArg y y0);Simpl;Auto.
Replace y1 with (ConsArg y y0);Simpl;Auto.
Elim H5;Intros.
Replace y2 with (ConsArg y y0);Simpl;Auto.
Elim H6;Intros;Simpl;Auto.
Save trans_SUP.

Goal (v,u,w:quasiterm)(SUB u v)->(SUB v w)->(SUB u w).
Unfold SUB;Intros;Apply trans_SUP with v;Auto.
Save trans_SUB.

Goal (v,u:quasiterm)(SUB u v)->~<quasiterm>u=v.
Intros.
Unfold not;Intros h;Cut (SUB u v);Auto;Elim h.
Elim u;Simpl;Auto.(*Case u=(V v) or u=(C f)*)
(*Case u=(Root f y)*)
Intros.
Elim H1;Intros.
Apply H0;Pattern y;Replace y with (Root f y);Auto.
Apply H0;Apply trans_SUB with (Root f y);Unfold SUB;Simpl;Auto.
(*Case u=(ConsArg y y0)*)
Intros.
Elim H2;Intros.
Apply H0;Pattern y;Replace y with (ConsArg y y0);Auto.
Elim H3;Intros.
Apply H1;Pattern y0;Replace y0 with (ConsArg y y0);Auto.
Elim H4;Intros.
Apply H0;Apply trans_SUB with (ConsArg y y0);Unfold SUB;Simpl;Auto.
Apply H1;Apply trans_SUB with (ConsArg y y0);Unfold SUB;Simpl;Auto.
Save SUB_diff.

Goal (t1,t2:quasiterm)(f:quasisubst)(SUB t1 t2)
     ->(SUB (Subst f t1) (Subst f t2)).
Unfold SUB;Induction t2;Simpl;Intros.
Elim H.
Elim H.
Elim H0;Intros h0.
Elim h0;Auto.
Auto.
Elim H1;Intros h1.
Elim h1;Auto.
Elim h1;Intros h2;Elim h2;Auto.
Save SUB_subst.

Goal (x:var)(t:quasiterm)(f:quasisubst)(IS_IN x t)
     ->(~<quasiterm>(V x)=t)->(SUB (f x) (Subst f t)).
Intros;Change (SUB (Subst f (V x)) (Subst f t));Apply SUB_subst.
Unfold SUB;Cut ~<quasiterm>(V x)=t;Auto;
Cut (IS_IN x t);Try Apply IS_IN_is_in;Auto.
Elim t.
(*Case t=(V v)*)
Simpl;Intros;Absurd <quasiterm>(V x)=(V v);Auto.
Elim H1;Auto.
(*Case t=(C f)*)
Simpl;Intros;Absurd False;Auto.
(*Case t=(Root f0 y)*)
Intros;Elim (quasiterm_eq_decP (V x) y);Intros.
```

```
Elim H4;Simpl;Auto.
Cut (IS_IN x (Root f0 y));Simpl;Auto.
(*Case t=(ConsArg y y0)*)
Simpl;Intros.
Elim (quasiterm_eq_decP (V x) y);Intros;Auto.
Elim (quasiterm_eq_decP (V x) y0);Intros;Auto.
Elim H3;Intros;Auto.
Save IS_IN_SUB.

(*********************************************************************)
(******************* Idempotence of a quasisubstitution **************)
(*********************************************************************)

Definition idempotent:quasisubst->Prop=
[s:quasisubst](x:var)<quasiterm>(s x)=(Subst s (s x)).

(*********************************************************************)
(************* If for each variable of the range *********************)
(******************** of a quasisubstitution, ************************)
(*************** this is transformed to itself ***********************)
(************* the quasisubstitution is idempotent. ******************)
(*********************************************************************)

Goal (f:quasisubst)((x:var)(range f x)-><quasiterm>(V x)=(f x))
     ->(idempotent f).
Unfold idempotent;Intros.
Elim (quasiterm_eq_decP (V x) (f x));Intros h.
(*Case <quasiterm>(V x)=(f x)*)
Elim h;Auto.
(*Case ~<quasiterm>(V x)=(f x)*)
Apply trans_equal with (Subst V (f x)).
Apply V_stab.
Apply eq_restriction_s_t.
Intros;Apply H;Apply range_init with x;Auto.
Save range_n_dom_idempotent.

(*********************************************************************)
(***** Conversely if a quasisubstitution is idempotent, then *********)
(***** the variables of the domain are not in the range. *************)
(*********************************************************************)

Goal (f:quasisubst)(idempotent f)->(x:var)(dom f x)->~(range f x).
Unfold not;Intros.
Elim H0;Elim H1;Intros.
Apply (eq_restriction_t_s (f y) V f);Auto.
Elim H;Elim V_stab with (f y);Auto.
Save idempotent_n_range.

(*********************************************************************)
(************* Version without range of the same result. *************)
(*********************************************************************)

Goal (f:quasisubst)(idempotent f)
     ->(x:var)(dom f x)->(y:var)~(IS_IN x (f y)).
Intros.
Elim (dom_decP f y);Intros
(*Case (dom f y)*)
Apply n_range_n_IS_IN;Auto.
Apply idempotent_n_range;Auto.
(*Case ~(dom f y)*)
```

```
Elim (n_dom f y H1);Simpl.
Unfold not;Intros h;Absurd (dom f y);Auto;Elim h;Auto.
Save idempotent_dom_n_IS_IN.

(*****************************************************************)
(**** If the quasisubstitutions f et g are such that: ************)
(***** 1)the quasisubstitutions f et g are idempotent. ***********)
(***** 2)the variables images under g are in f(t), ***************)
(***** 3)the variables of the domain of g are in f(t), ************)
(**** then g o f est idempotent. *********************************)
(*****************************************************************)

Goal (t:quasiterm)(f,g:quasisubst)(idempotent f)->(idempotent g)
     ->(under g (Subst f t))->(over g (Subst f t))
     ->(idempotent [x:var](Subst g (f x))).
Unfold under;Unfold over;Intros;Apply range_n_dom_idempotent;Intros.
Elim H3;Intros.
Elim (dom_decP g x);Intros.
(*Case (dom g x)*)
Absurd (IS_IN x (Subst g (f y)));Auto.
Apply n_IS_IN_Subst.
Intros;Apply idempotent_dom_n_IS_IN;Auto.
(*Case ~(dom g x)*)
Elim (dom_decP f x);Intros.
(*Case ~(dom g x);(dom f x)*)
Absurd (IS_IN x (f y)).
Apply idempotent_dom_n_IS_IN;Auto.
Apply (n_range_IS_IN_Subst g (f y));Auto.
Unfold not;Intros H8;Elim H8;Intros.
Absurd (IS_IN x (Subst f t)).
Apply n_IS_IN_Subst.
Intros;Apply idempotent_dom_n_IS_IN;Auto.
Apply (H1 x y0);Auto.
(*Case ~(dom g x);~(dom f x)*)
Elim (n_dom f x H7);Simpl;Elim (n_dom g x H6);Simpl;Auto.
Save idempotent_Fondamental.

(*****************************************************************)
(************** Two evaluations of g(f(t)) : *********************)
(*****************************************************************)

Goal (f,g:quasisubst)(t:quasiterm)
  <quasiterm>(Subst g (Subst f t))=(Subst [x:var](Subst g (f x)) t).
Induction t;Simpl;Intros;Auto.
Elim H;Auto.
Elim H;Elim H0;Auto.
Save comp_subst.

Goal (f,g:quasisubst)(t:quasiterm)
<quasiterm>(Subst [x:var](Subst g (f x)) t)=(Subst g (Subst f t)).
Intros;Symmetry;Apply comp_subst.
Save exp_comp_subst.

(*****************************************************************)
(******** First lemma of induction ********************************)
(****** if the variables of the domain of f and the range of f ***********)
(*** are in (ConsArg p1 p2) and if the variables of the domain of g ****)
(****** are in f(ConsArg p3 p4) then the variables of the domain ******)
(***** of g o f are in (ConsArg (ConsArg p1 p3) (ConsArg p2 p4)). ***)
(*****************************************************************)
```

```
Goal (p1,p2,p3,p4:quasiterm)(f,g:quasisubst)(over f (ConsArg p1 p2))
     ->(over g (ConsArg (Subst f p3) (Subst f p4)))
     ->(under f (ConsArg p1 p2))
     ->(over [x:var](Subst g (f x))
(ConsArg (ConsArg p1 p3) (ConsArg p2 p4))).
Unfold over under;Intros.
Cut ~(IS_IN x (ConsArg p1 p2)).
Intros h;Elim (H x h).
Simpl;Apply H0;Unfold not;Intros;Elim H2.                                    970

(*n_range_IS_IN_Subst:(f:quasisubst)(t:quasiterm)(x:var)
(~(range f x))->(IS_IN x (Subst f t))->(IS_IN x t)*)

Apply n_range_IS_IN_Subst with f.
Unfold not;Intros h0;Elim h0:Intros.
Elim h;Apply H1 with y;Auto.
Simpl;Elim H3;Auto.
Simpl;Unfold not;Intros h;Elim H2;Simpl;Elim h;Intros;Auto.
Save over_comp.                                                             980

(*********************************************************************)
(******** Second lemma of induction ********************************)
(******* If the variables of the domain of f and the images under f ****)
(******* are in (ConsArg p1 p2) and if *****************************)
(******* the variables of the domain of g and the images under g*******)
(******* are in f(ConsArg p3 p4) then the images under gof ***********)
(******* are in (ConsArg (ConsArg p1 p3) (ConsArg p2 p4)). **********)
(*********************************************************************)
                                                                            990
Goal (p1,p2,p3,p4:quasiterm)(f,g:quasisubst)
     (under f (ConsArg p1 p2))
     ->(under g (ConsArg (Subst f p3) (Subst f p4)))
     ->(over f (ConsArg p1 p2))
     ->(over g (ConsArg (Subst f p3) (Subst f p4)))
     ->(under [x:var](Subst g (f x))
          (ConsArg (ConsArg p1 p3) (ConsArg p2 p4))).
Unfold over dom under;Intros.

(*under_IS_IN_Subst_IS_IN:(f:quasisubst)(t:quasiterm)                       1000
(under f t)->(x:var)(IS_IN x (Subst f t))->(IS_IN x t)*)

Apply under_IS_IN_Subst_IS_IN with g.
(*Goal (under...)*)
Unfold under over dom;Intros.
Cut (IS_IN x0 (Subst f (ConsArg p3 p4))).
Intros H7;Elim (IS_IN_Subst_exi_IS_IN f (ConsArg p3 p4) x0 H7);Intros.
Elim H8:Intros h1 h2.
Apply under_IS_IN_Subst_IS_IN with f.
Unfold under;Intros.                                                        1010
Elim H with x2 y1:Simpl:Auto.
Elim (H0 x0 y0):Simpl:Auto.
Elim (H0 x0 y0):Simpl:Auto.
(*Goal (IS_IN x...)*)
Elim (IS_IN_Subst_exi_IS_IN g (f y) x H4):Intros.
Elim H5:Intros.
Apply IS_IN_Subst_IS_IN with x0:Auto.
Elim (dom_decP f y):Intros.
(**)
Elim (H x0 y):Simpl:Auto.                                                   1020
```

```
Cut (IS_IN x0 (f y));Auto;Elim (n_dom f y H8);Intros.
Elim (dom_decP g x0);Intros.
(**)
Elim (IS_IN_decP x0 (ConsArg (Subst f p3) (Subst f p4)));Intros h.
Elim (IS_IN_Subst_exi_IS_IN f (ConsArg p3 p4) x0 h);Intros x1 h1;
Elim h1;Intros.
Elim (dom_decP f x1);Intros.
(**)
Elim H with x0 x1;Simpl;Auto.
(**)                                                                    1030
Cut (IS_IN x0 (f x1));Auto;Elim (n_dom f x1 H13);Intros.
Cut (IS_IN x1 (ConsArg p3 p4));Auto;Elim H14.
Simpl;Intros h2;Elim h2;Intros;Auto.
Absurd <quasiterm>(V x0)=(g x0);Auto.
Absurd <quasiterm>(V y)=(Subst g (f y));Auto.
Elim (n_dom f y H8);Simpl;Elim H9;Elim (n_dom g x0 H10);Auto.
Save under_comp.

(*********************************************************************)
(*** If the domain of the quasisubstitution f is finite, ***********)                1040
(*** then either f is the injection V from var is_ino quasiterm, ********)
(******** or there exists x in var such that f(x) is not V(x). **********)
(*** In the second case f decreases the number of variables. **********)
(*********************************************************************)

Goal (t:quasiterm)(f:quasisubst)(over f t)
     ->((<var>Ex([x:var]
     (~<quasiterm>(V x)=(f x))))\/((v:var)<quasiterm>(V v)=(f v))).
(*cut*)
Cut (l:listv)(f:quasisubst)                                            1050
((x:var)(~<quasiterm>(V x)=(f x))->(IS_IN_LV x l))
->((<var>Ex([x:var](~<quasiterm>(V x)=(f x))))\/
((v:var)<quasiterm>(V v)=(f v))).
Unfold over;Intros.
Apply H with (list_var t);Intros.
Apply IS_IN_IS_IN_LV.
Elim (IS_IN_decP x t);Intros;Auto.
Absurd <quasiterm>(V x)=(f x);Auto.
(*proof of the cut*)
Induction l:Simpl:Intros.                                              1060
Right;Intros.
Elim (quasiterm_eq_decP (V v) (f v));Intros:Auto.
Elim (H v H0).
Elim (quasiterm_eq_decP (V v) (f v));Intros.
Apply (H f);Intros.
Elim (H0 x H2);Intros:Auto.
Absurd <quasiterm>(V v)=(f v);Auto;Elim H3:Auto.
Left:Exists v:Auto.
Save ident_or_not.
                                                                       1070
Goal (t:quasiterm)(f:quasisubst)(over f t)
     ->{x:var|~<quasiterm>(V x)=(f x)}+{(x:var)<quasiterm>(V x)=(f x)}.
Cut (l:listv)(f:quasisubst)
((x:var)(~<quasiterm>(V x)=(f x))->(IS_IN_LV x l))
->{x:var|~<quasiterm>(V x)=(f x)}+{(x:var)<quasiterm>(V x)=(f x)}.
Unfold over:Intros.
Apply H with (list_var t):Intros.
Apply IS_IN_IS_IN_LV.
Elim (IS_IN_decP x t):Intros:Auto.
Absurd <quasiterm>(V x)=(f x):Auto.                                    1080
```

```
(**)
Induction l;Simpl;Intros.
Right;Intros.
Elim (quasiterm_eq_decP (V x) (f x));Intros;Auto.
Elim (H x H0).
Elim (quasiterm_eq_decS (V v) (f v));Intros.
Apply (H f);Intros.
Elim (H0 x H1);Intros;Auto.
Absurd <quasiterm>(V v)=(f v);Auto;Elim H2;Auto.
Left;Exists v;Auto.                                                    1090
Save ident_or_notS.

(*********************************************************************)
(******* If f is idempotent and if the variables of the domain of f ****)
(******* and the images under f are in (ConsArg t1 t3) **************)
(******* and if the domain of f is not empty ************************)
(***** then the number of different variables in ConsArg(f(t2),f(t4)) ***)
(***** is strictly less than in ConsArg(ConsArg(t1,t2),ConsArg(t3,t4)).**)
(*********************************************************************)

                                                                       1100
Goal (t1,t2,t3,t4:quasiterm)(n:nat)
     (DIFFELNB (list_var (ConsArg (ConsArg t1 t2)(ConsArg t3 t4))) n)
     ->(f:quasisubst)(idempotent f)
     ->(over f (ConsArg t1 t3))
     ->(under f (ConsArg t1 t3))
     -><var>Ex([x:var](~<quasiterm>(V x)=(f x)))
     ->(n0:nat)
         (DIFFELNB (list_var (ConsArg (Subst f t2) (Subst f t4))) n0)
     ->(le (S n0) n).
Unfold over under;Intros.                                              1110
Elim H3;Intros.
Apply DIFFELNB_st_inclv_le_S with
(list_var (ConsArg (Subst f t2) (Subst f t4)))
(list_var (ConsArg (ConsArg t1 t2)(ConsArg t3 t4)));Auto.
Apply st_inclv_init with x.
Change ~(IS_IN_LV x (list_var (Subst f (ConsArg t2 t4)))).
Unfold not;Intros;Cut (IS_IN x (Subst f (ConsArg t2 t4)));Intros.
2:Apply IS_IN_LV_IS_IN;Auto.
Elim (IS_IN_Subst_exi_IS_IN f (ConsArg t2 t4) x H7);Intros.
Elim H8;Intros.                                                        1120
Absurd (IS_IN x (f x0));Auto;Apply idempotent_dom_n_IS_IN;Auto.
Apply IS_IN_IS_IN_LV;Elim (IS_IN_decP x (ConsArg t1 t3));Intros.
Cut (IS_IN x (ConsArg t1 t3)).
2:Auto.
Simpl;Intros h;Elim h;Auto.
Absurd <quasiterm>(V x)=(f x);Auto.
Intros;Cut (IS_IN y (Subst f (ConsArg t2 t4))).
2:Apply IS_IN_LV_IS_IN;Auto.
Intros;Apply IS_IN_IS_IN_LV;
Elim (IS_IN_Subst_exi_IS_IN f (ConsArg t2 t4) y H7);Intros.            1130
Elim H8;Intros.
Elim (dom_decP f x0);Intros.
Cut (IS_IN y (ConsArg t1 t3)).
2:Apply (H2 y x0);Auto.
Simpl;Intros h;Elim h;Auto.
Cut <quasiterm>(V x0)=(f x0).
2:Apply n_dom;Auto.
Intros;Cut <var>y=x0.
Intros h;Cut (IS_IN x0 (ConsArg t2 t4)).
2:Auto.                                                                1140
```

```
Elim h;Simpl;Intros h1;Elim h1;Auto.
Cut (IS_IN y (f x0));Auto;Elim H12;Simpl;Auto.
Save f_n_id_minus.

(*************************************************************************)
(*************** The iterative construction of the unifiers. ***************)
(****** Here is a method which does not requires the axiom of choice *******)
(***************** but uses an induction over the integers. ****************)
(*************************************************************************)

                                                                        1150
Inductive Definition elem_subst[x:var;t:quasiterm;f:quasisubst]:Prop=
elem_subst_init:((y:var)(~<var>x=y)->><quasiterm>(V y)=(f y))
     ->((y:var)(<var>x=y)->><quasiterm>t=(f y))->(elem_subst x t f).

Goal (t:quasiterm)(n:var)(f:quasisubst)
     {g:quasisubst|(<quasiterm>t=(g n))
     /\((p:nat)(~<nat>n=p)->><quasiterm>(f p)=(g p))}.
Induction n.
Intros;Exists [m:nat](<quasiterm>Match m
with t [p:nat][b:quasiterm](f m));Split;Auto.                           1160
Induction p;Auto.
Intros;Absurd <nat>O=O;Auto.
Intros.
Elim (H [x:nat](f (S x)));Intros g0 h0;Elim h0;Intros.
Exists [m:nat](<quasiterm>Match m
with (f O) [p:nat][b:quasiterm](g0 p));Split;Auto.
Induction p;Auto.
Intros y0 h1 h2;Apply H1;Unfold not;Intros;Elim h2;Auto.
Save sig_elem_subst0.

                                                                        1170
Goal (x:var)(t:quasiterm){f:quasisubst|(elem_subst x t f)}.
Intros;Elim (sig_elem_subst0 t x V);Intros.
Exists x0;Elim p;Intros;Apply elem_subst_init;Auto.
Intros y0 h;Elim h;Auto.
Save sig_elem_subst.

Goal (t:quasiterm)(x:var)(f:quasisubst)((y:var)
     ((~(<var>x=y))->><quasiterm>(V y)=(f y)))
     ->(~(IS_IN x t))->><quasiterm>t=(Subst f t).
Intros.                                                                 1180
Replace (Subst f t) with (Subst V t);Try Apply V_stab.
Apply eq_restriction_s_t.
Intros;Elim (var_eq_decP x x0);Intros;Auto.
Absurd (IS_IN x0 t);Auto;Elim H2;Auto.
Save elem_subst_conserve.

(*************************************************************************)
(************** Order in quasisubst ************************************)
(*************************************************************************)

                                                                        1190
Inductive Definition less_subst[f,g:quasisubst]:Prop=
less_subst_init:(h:quasisubst)((x:var)(<quasiterm>(Subst h (f x))=(g x)))
     ->(less_subst f g).

(*************************************************************************)
(********************** Definition of an unifier: ************************)
(*************************************************************************)

Definition unif:quasisubst->quasiterm->quasiterm->Prop
=[f:quasisubst][t,u:quasiterm]<quasiterm>(Subst f t)=(Subst f u).        1200
```

```
(*****************************************************************)
(***************** Definition of a minimal unifier: ****************)
(*****************************************************************)

Definition min_unif:quasisubst->quasiterm->quasiterm->Prop
=[f:quasisubst][t,u:quasiterm](g:quasisubst)(unif g t u)
     ->(less_subst f g).

(*****************************************************************)                1210
(***************** Specification of the problem of the unification ******)
(*****************************************************************)
(********************** Unif defines the minimal unifiers.*************)
(*****************************************************************)

Inductive Definition Unification[t1,t2:quasiterm]:Set=
Unif_succeed:(f:quasisubst)
     (unif f t1 t2)
     ->(idempotent f)
     ->(over f (ConsArg t1 t2))                                                  1220
     ->(under f (ConsArg t1 t2))
     ->(min_unif f t1 t2)
(*------------- Verifications on the terms:begin --------------*)
(**) ->((l:list_nat)(L_TERM l t1)->(L_TERM l t2)
(**)      ->(<nat>(Length t1)=(Length t2))->(term_subst l f))
(*------------- Verifications on the terms:end ----------------*)
     ->(Unification t1 t2)
|Unif_fail:((f:quasisubst)~<quasiterm>(Subst f t1)=(Subst f t2))
     ->(Unification t1 t2).
                                                                                 1230
(*****************************************************************)
(************** Failure in the unification with the head symbol: *******)
(*****************************************************************)

Inductive Definition head_diff:quasiterm->quasiterm->Prop=
Fail_hd1:(l:fun)(t1,t2:quasiterm)(head_diff (C l) (ConsArg t1 t2))
|Fail_hd2:(l:fun)(t1,t2:quasiterm)(head_diff (ConsArg t1 t2) (C l))
|Fail_hd3:(l,l0:fun)(t:quasiterm)(head_diff (C l) (Root l0 t))
|Fail_hd4:(l,l0:fun)(t:quasiterm)(head_diff (Root l0 t) (C l))
|Fail_hd5:(t1,t2,t3:quasiterm)(l:fun)                                            1240
          (head_diff (ConsArg t1 t2)(Root l t3))
|Fail_hd6:(t1,t2,t3:quasiterm)(l:fun)
          (head_diff (Root l t3)(ConsArg t1 t2)).

Goal (t1,t2:quasiterm)(head_diff t1 t2)->(Unification t1 t2).
Intros;Apply Unif_fail;Elim H;Intros.
Apply Diff with BC;Simpl;Auto.
Apply Diff with BConsArg;Simpl;Auto.
Apply Diff with BC;Simpl;Auto.
Apply Diff with BRoot;Simpl;Auto.                                                1250
Apply Diff with BConsArg;Simpl;Auto.
Apply Diff with BRoot;Simpl;Auto.
Save Decomp_fail.

Hint Fail_hd1 Fail_hd2 Fail_hd3 Fail_hd4 Fail_hd5 Fail_hd6 Decomp_fail.

(*****************************************************************)
(*************** Symetry of the problem of the unification **********)
(*****************************************************************)
                                                                                 1260
```

```
Goal (t1,t2:quasiterm)(Unification t1 t2)->(Unification t2 t1).
Intros;Elim H;Unfold idempotent over under min_unif unif;Intros.
Apply Unif_succeed with f;
Unfold idempotent over under min_unif unif;Auto;Intros.
Apply o;Unfold not;Intros h;Elim H0;Simpl;Elim h;Auto.
Simpl;Elim (u0 x y);Auto.
Apply Unif_fail;Intros.
Unfold not;Intros;Elim (n f);Auto.
Save sym_Unification.

(*********************************************************************)
(********** Unification when one of the terms is a variable : ********)
(*********************************************************************)

Goal (n:var)(t:quasiterm)(Unification (V n) t).
Intros;Elim (IS_IN_decS n t);Intros.
Elim (quasiterm_eq_decS (V n) t);Intros.
(*Case (IS_IN n t) and <quasiterm>(V n)=t*)
Elim a0;Apply Unif_succeed with [x:var](V x);
Unfold unif idempotent over under;Simpl;Auto;Intros.
Absurd <quasiterm>(V y)=(V y);Auto.
Unfold min_unif unif;Intros.
Apply less_subst_init with g;Auto.
(*------------- Verifications on the terms:begin --------------*)
Apply term_subst_init;Intros;Apply term_init;Simpl;Auto.
(*------------- Verifications on the terms:end ----------------*)
(*Case (IS_IN n t) and ~<quasiterm>(V n)=t*)
Apply Unif_fail;Intros;Apply SUB_diff.
Simpl;Apply IS_IN_SUB;Auto.
(*Case ~(IS_IN n t)*)
Elim (sig_elem_subst n t);Intros.
Apply Unif_succeed with x;
Unfold unif idempotent over under;Simpl;Elim p;Intros.
Replace (x n) with t;Auto.
Apply elem_subst_conserve with n;Auto.
Elim (var_eq_decP n x0);Intros.
Elim (H0 x0);Auto.
Apply elem_subst_conserve with n;Auto.
Elim (H x0);Simpl;Auto.
Elim (var_eq_decP x0 n);Intros.
Elim H1;Auto.
Elim (H x0);Auto.
Unfold not;Intros;Elim H2;Auto.
Elim (var_eq_decP n y);Intros.
Replace t with (x y);Auto.
Symmetry;Auto.
Absurd <quasiterm>(V y)=(x y);Auto.
Unfold min_unif unif;Intros.
Apply less_subst_init with g.
Intros;Elim (var_eq_decP n x0);Intros.
Elim H2;Elim (H0 n);Auto.
Elim (H x0);Auto.
(*------------- Verifications on the terms:begin --------------*)
Apply term_subst_init.
Intros;Elim (var_eq_decP x0 n);Intros.
Elim (H0 x0);Auto.
Apply Length_SO_term;Auto.
Elim (H x0).
Apply term_init;Simpl;Auto.
Unfold not;Intros;Elim H4;Auto.
```

```
(*------------ Verifications on the terms:end --------------*)
Save UnifV1.

Goal (t:quasiterm)(x:var)(Unification t (V x)).
Intros;Apply sym_Unification;Apply UnifV1.
Save UnifV2.

Hint UnifV1 UnifV2.

(*****************************************************************)
(********** Unification when one of the terms is a constant : ********)
(*****************************************************************)

Goal (t:quasiterm)(l:fun)(Unification (C l) t).
Induction t;Auto.
Intros;Elim (fun_eq_decS l f);Intros.
Elim a;Apply Unif_succeed with V;
Unfold unif idempotent over under;Simpl;Auto.
Intros;Absurd <quasiterm>(V y)=(V y);Auto.
Unfold min_unif unif;Intros.
Apply less_subst_init with g ;Auto.
(*------------ Verifications on the terms:begin --------------*)
Intros;Apply term_subst_init;Intros;Apply term_init;Simpl;Auto.
(*------------ Verifications on the terms:end --------------*)
Apply Unif_fail;Intros;Simpl.
Apply C_diff_C;Auto.
Save UnifC1.
Goal (t:quasiterm)(l:fun)(Unification t (C l)).
Intros;Apply sym_Unification;Apply UnifC1.
Save UnifC2.

Hint UnifC1 UnifC2.

(*****************************************************************)
(********** Link between the unification of  t1 and t2 ***************)
(****** and the unification of (Root l1 t1) and (Root l2 t2) : *********)
(*****************************************************************)

Goal (t1,t2:quasiterm)(l1,l2:fun)
     (Unification t1 t2)
     ->(Unification (Root l1 t1) (Root l2 t2)).
Intros;Elim (fun_eq_decS l1 l2);Intros.
Elim a:Elim H;Unfold unif idempotent over under;Intros.
Apply Unif_succeed with f;
Unfold unif idempotent over under;Simpl;Auto.
Elim u;Auto.
Unfold min_unif unif;Simpl;Intros.
Apply m;Unfold unif.
Apply proj_Root2 with l1 l1;Auto.
Intros.
Elim H0:Elim H1;Intros;Apply t;Auto.
Elim H4;Auto.
Apply Unif_fail;Intros.
Simpl;Apply Root_diff_Root;Auto.
Apply Unif_fail;Intros.
Simpl;Apply Root_diff_Root;Auto
Save UnifRoot.

Hint UnifRoot.
```

```
(*************************************************************************)
(********** The failure of the unification of t1 and t3 implies ***********)
(* the failure of the unification of (ConsArg t1 t2) and (ConsArg t3 t4) : **)
(*************************************************************************)

Goal (t1,t2,t3,t4:quasiterm)
     ((f:quasisubst)(~<quasiterm>(Subst f t1)=(Subst f t3)))
     ->(Unification (ConsArg t1 t2) (ConsArg t3 t4)).
Intros;Apply Unif_fail;Intros.
Simpl;Apply ConsArg_diff_ConsArg;Auto.                                  1390
Save UnifConsArgfail1.

Hint UnifConsArgfail1.

(*************************************************************************)
(********** The unification of the ground terms : ********************)
(*************************************************************************)

Goal (t1,t2:quasiterm)
     (DIFFELNB (list_var (ConsArg t1 t2)) O)                            1400
     ->(Unification t1 t2).
Intros;Elim (quasiterm_eq_decS t1 t2);Intros.
Elim a.
Apply Unif_succeed with V;
Unfold unif idempotent over under min_unif;Auto.
Intros.
Absurd <quasiterm>(V y)=(V y);Auto.
Intros;Apply less_subst_init with g;Auto.
Intros;Apply term_subst_init;Intros;Apply term_init;Simpl;Auto.
Apply Unif_fail;Intros.                                                 1410
Elim (clossubst t1 f).
Elim (clossubst t2 f);Auto.
Apply closConsArg2 with t1;Apply DIFFELNB_O_clos;Auto.
Apply closConsArg1 with t2;Apply DIFFELNB_O_clos;Auto.
Save Unif_DIFFELNB_O.

(*************************************************************************)
(********** The number of variables of (ConsArg t1 t3)  **************)
(************ and the one of the variables of (ConsArg t2 t4) *********)
(********** are less or equal to the number of the variables of *********)  1420
(********** (ConsArg (ConsArg t1 t2)(ConsArg t3 t4)). **************)
(*************************************************************************)

Goal (t1,t2,t3,t4:quasiterm)(n:nat)
     (DIFFELNB (list_var (ConsArg (ConsArg t1 t2)(ConsArg t3 t4))) n)
     ->(n0:nat)(DIFFELNB (list_var (ConsArg t2 t4)) n0)
     ->({(le (S n0) n)}+{<nat>n0=n}).
Intros;Elim (le_decS n0 n);Intros.
Apply (le_S_eqS n0 n a).
Absurd (le n0 n);Auto.                                                  1430
Apply inclv_le with (list_var (ConsArg t2 t4))
(list_var (ConsArg (ConsArg t1 t2) (ConsArg t3 t4)));Auto.
Apply inclv_init;Intros;Apply IS_IN_IS_IN_LV;Simpl.
Elim (IS_IN_LV_IS_IN (ConsArg t2 t4) y H1);Auto.
Save DIFFELNB_ConsArg_ConsArg24_le.

Goal (t1,t2,t3,t4:quasiterm)(n:nat)
     (DIFFELNB (list_var (ConsArg (ConsArg t1 t2) (ConsArg t3 t4))) n)
     ->(n0:nat)(DIFFELNB (list_var (ConsArg t1 t3)) n0)
     ->({(le (S n0) n)}+{<nat>n0=n}).                                   1440
```

```
Intros;Elim (le_decS n0 n);Intros h.
Apply (le_S_eqS n0 n h).
Absurd (le n0 n);Auto.
Apply inclv_le with (list_var (ConsArg t1 t3))
(list_var (ConsArg (ConsArg t1 t2) (ConsArg t3 t4)));Auto.
Apply inclv_init;Intros;Apply IS_IN_IS_IN_LV;Simpl.
Elim (IS_IN_LV_IS_IN (ConsArg t1 t3) y H1);Auto.
Save DIFFELNB_ConsArg_ConsArg13_le.

Hint eq_V_stab.                                                          1450

(*****************************************************************)
(******************** If f unifies t1 and u1 *********************)
(********** and if g unifies (Subst f t2) (Subst f u2), **********)
(************** then [x:var](Subst g (f x)) unifies ***************)
(************** (ConsArg t1 t2) et (ConsArg u1 u2). **********)
(*****************************************************************)

Goal (t1,t2,u1,u2:quasiterm)(f,g:quasisubst)
     (unif f t1 u1)                                                      1460
     ->(unif g (Subst f t2) (Subst f u2))
     ->(unif [x:var](Subst g (f x)) (ConsArg t1 t2) (ConsArg u1 u2)).
Unfold unif;Simpl;Intros.
Elim (comp_subst f g t2);Elim (comp_subst f g u2);
Elim (comp_subst f g t1);Elim (comp_subst f g u1).
Elim H;Elim H0;Auto.
Save unif_comp.

Hint unif_comp.

                                                                         1470
(*****************************************************************)
(***************** If f is a minimal unifier of t1 and u1 ************)
(********** and if g is a one of (Subst f t2) (Subst f u2), then *******)
(********************** [x:var](Subst g (f x)) ***********************)
(************* is a one of (ConsArg t1 t2) and (ConsArg u1 u2) : ****)
(*****************************************************************)

Goal (t1,t2,u1,u2:quasiterm)(f,g:quasisubst)
     (min_unif f t1 u1)
     ->(min_unif g (Subst f t2) (Subst f u2))                            1480
     ->(min_unif [x:var](Subst g (f x)) (ConsArg t1 t2) (ConsArg u1 u2)).
Unfold min_unif unif;Simpl;Intros.
Elim (H g0);Intros.
Elim (H0 h);Intros.
Apply less_subst_init with h0.
Intros;Elim (H2 x).
Elim (exp_comp_subst g h0 (f x)).
Apply eq_restriction_s_t;Auto.
Elim (exp_comp_subst f h t2);Elim (exp_comp_subst f h u2).
Transitivity (Subst g0 t2).                                              1490
Apply eq_restriction_s_t;Auto.
Transitivity (Subst g0 u2).
Apply proj_ConsArg2 with (Subst g0 t1) (Subst g0 u1);Auto;Intros.
Apply eq_restriction_s_t;Auto.
Apply proj_ConsArg1 with (Subst g0 t2) (Subst g0 u2);Auto;Intros.
Save min_unif_comp.

Hint min_unif_comp.

(*------------ Verifications on the terms:begin ------------*)           1500
```

```
Goal (t1,t2,t3,t4:quasiterm)(f,g:quasisubst)(l:list_nat)
     (L_TERM l (ConsArg t1 t2))
     ->(L_TERM l (ConsArg t3 t4))
     ->(<nat>(Length (ConsArg t1 t2))=(Length (ConsArg t3 t4)))
     ->((l:list_nat)(L_TERM l t1)->(L_TERM l t3)
         ->(<nat>(Length t1)=(Length t3))->(term_subst l f))
     ->((l:list_nat)(L_TERM l (Subst f t2))->(L_TERM l (Subst f t4))
         ->(<nat>(Length (Subst f t2))=(Length (Subst f t4)))
         ->(term_subst l g))
     ->(term_subst l [x:var](Subst g (f x))).
Simpl;Intros.
Elim H;Elim H0;Intros.
Elim H5;Elim H7;Intros.
Cut (term_subst l f).
Intros hyp;Apply term_subst_init.
Intros;Simpl;Apply term_term_subst.
Elim hyp;Auto.
Apply H3;Try Apply L_TERM_term_subst;Auto.
Elim (term_subst_eq_Length l f t2);Elim (term_subst_eq_Length l f t4);
Auto.
Cut <nat >(<nat>Match (Length t1) with (Length t2) [x:nat]S)
=(<nat>Match (Length t3) with (Length t4) [x:nat]S);Auto.
Replace (Length t1) with (S O);Try Apply SIMPLE_SO;Auto.
Replace (Length t3) with (S O);Try Apply SIMPLE_SO;Auto.
Apply (H2 l H8 H10).
Replace (Length t1) with (S O);Try Apply SIMPLE_SO;Auto.
Save term_subst_comp.
(*------------ Verifications on the terms:end ---------------*)

(*********************************************************************)
(********** If f is a minimal unifier of t1 and t3 ******************)
(********** whose domain and image are in (ConsArg t1 t3) *********)
(***************** and if g is a minimal unifier *********************)
(***************** of (Subst f t2) and (Subst f  t4) ******************)
(******** then [x:var](Subst g (f x)) is also a minimal unifier *********)
(*********** of (ConsArg t1 t2) and (ConsArg t3 t4) : ***************)
(*********************************************************************)

Hint over_comp under_comp.

Goal (t1,t2,t3,t4:quasiterm)(f,g:quasisubst)
     (unif f t1 t3)
     ->(idempotent f)
     ->(over f (ConsArg t1 t3))
     ->(under f (ConsArg t1 t3))
     ->(min_unif f t1 t3)
     ->((l:list_nat)(L_TERM l t1)->(L_TERM l t3)
         ->(<nat>(Length t1)=(Length t3))->(term_subst l f))
     ->(unif g (Subst f t2) (Subst f t4))
     ->(idempotent g)
     ->(over g (ConsArg (Subst f t2) (Subst f t4)))
     ->(under g (ConsArg (Subst f t2) (Subst f t4)))
     ->(min_unif g (Subst f t2) (Subst f t4))
     ->((l:list_nat)(L_TERM l (Subst f t2))->(L_TERM l (Subst f t4))
         ->(<nat>(Length (Subst f t2))=(Length (Subst f t4)))
         ->(term_subst l g))
     ->(Unification (ConsArg t1 t2) (ConsArg t3 t4)).
Intros;Apply Unif_succeed with [x:var](Subst g (f x));Intros;Auto.
```

```
(*Apply unif_comp;over_comp;under_comp;min_unif_comp*)
Apply (idempotent_Fondamental (ConsArg t2 t4));Auto.
(*------------- Verifications on the terms:begin --------------*)
Apply term_subst_comp with t1 t2 t3 t4;Auto.
(*------------- Verifications on the terms:end ----------------*)
Save two_succes.

(*****************************************************************)
(************** If f is a minimal unifier of t1 and t3 ***************)
(************ whose domain and image are in (ConsArg t1 t3) ********)
(********************** and if the unification  ********************)
(****************** of (Subst f t2) and (Subst f  t4) fails **************)
(**** then the unification of (ConsArg t1 t2) and (ConsArg t3 t4) *****)
(***************************** fails also. ***************************)
(*****************************************************************)

Goal (t1,t2,t3,t4:quasiterm)(f:quasisubst)
     (unif f t1 t3)
     ->(idempotent f)
     ->(min_unif f t1 t3)
     ->((g:quasisubst)~<quasiterm>(Subst g (Subst f t2))
                                  =(Subst g (Subst f t4)))(*1*)
     ->(Unification (ConsArg t1 t2) (ConsArg t3 t4)).
Unfold unif idempotent over under min_unif;Intros;Apply Unif_fail;Intros.
Unfold not;Simpl;Intros.
Elim (H2 f0).
Elim (H1 f0);Unfold unif;Intros.
(**)
Elim (exp_comp_subst f f0 t2);Elim (exp_comp_subst f f0 t4).
Replace (Subst [x:var](Subst f0 (f x)) t2) with (Subst f0 t2).
Replace (Subst [x:var](Subst f0 (f x)) t4) with (Subst f0 t4).
Apply proj_ConsArg2 with (Subst f0 t1) (Subst f0 t3);Auto.
(**)
Apply eq_restriction_s_t;Intros.
Elim (H4 x).
Replace (Subst h (f x)) with (Subst h (Subst f (f x))).
Elim (exp_comp_subst f h (f x));Apply eq_restriction_s_t;Auto.
Elim H0;Auto.
(**)
Apply eq_restriction_s_t;Intros.
Elim (H4 x).
Replace (Subst h (f x)) with (Subst h (Subst f (f x))).
Elim (exp_comp_subst f h (f x));Apply eq_restriction_s_t;Auto.
Elim H0;Auto.
(**)
Apply proj_ConsArg1 with (Subst f0 t2) (Subst f0 t4);Auto.
Save one_only_succes.

(*****************************************************************)
(************ If t1 and t3 can be unified by the identity, ***********)
(******** then (ConsArg t1 t2) and (ConsArg t3 t4) can this also *****)
(**************** if and only if t2 and t4 can this . ***************)
(*****************************************************************)

Goal (t1,t2,t3,t4:quasiterm)(f:quasisubst)
     ((x:var)<quasiterm>(V x)=(f x))
     ->(unif f t1 t3)
     ->(Unification t2 t4)
     ->(Unification (ConsArg t1 t2) (ConsArg t3 t4))
Unfold unif;Intros.
```

```
Elim H1;Unfold unif over under idempotent min_unif;Intros.
(*Success*)
Apply two_succes with f f0;
Replace (Subst f t2) with t2;
Replace (Subst f t4) with t4;
Unfold unif over under idempotent dom;Simpl;Auto.
(*Apply eq_V_stab*)
Intros;Absurd <quasiterm>(V y)=(f y);Auto.
Unfold min_unif;Intros.
Apply less_subst_init with g.                                          1630
Intros;Elim (H x);Auto.
(*------------- Verifications on the terms:begin -------------*)
(**)Intros;Apply term_subst_init;Intros;Apply term_init;
(**)Elim H;Simpl;Auto.
(*------------- Verifications on the terms:end ---------------*)
(*Fail*)
Apply Unif_fail.
Simpl;Intros;Unfold not;Intros.
Elim (n f0);
Apply proj_ConsArg2 with (Subst f0 t1) (Subst f0 t3);Auto.             1640
Save eq_V_stab3.

(*****************************************************************)
(******************** For any terms u1 and u2, *******************)
(********** either there exists a minimal unifier of u1 and u2 ********)
(*********** whose domain and image are in (ConsArg u1 u2) *******)
(************** or u1 and u2 do not admit any unifier. *************)
(*****************************************************************)

Goal (n:nat)(u1,u2:quasiterm)                                          1650
     (DIFFELNB (list_var (ConsArg u1 u2)) n)
     ->(Unification u1 u2).
Intros n;Pattern n;Apply (ind_leS n).


   (*************************************************************)
   (********************* Case without variable : ***************)
   (*************************************************************)
Intros;Apply Unif_DIFFELNB_O;Auto.
   (*************************************************************)   1660
   (************** Case with p+1 different variables, ***************)
   (***************** induction on u1, then on u2 : *****************)
   (*************************************************************)
Induction u1;Auto.(*Apply UnifV1;UnifC1*)
(*Case u1 = (Root ...) : *)
Induction u2;Auto.(*Apply UnifV2;UnifC2;UnifRoot;Decomp_fail;Fail_hd6*)
(*Case u1 = (ConsArg ...) : *)
Induction u2;Auto.(*Apply UnifV2;UnifC2;Decomp_fail;Fail_hd5*)
Intros.
   (**       1 subgoal                                          *)   1670
   (**       (Unification (ConsArg y y0) (ConsArg y1 y2))       *)
   (**        ============================                      *)
   (**       number of different variables in (ConsArg y y1) :  *)
Elim (DIFFELNBor (list_var (ConsArg y y1)));Intros p0 H5.

   (*************************************************************)
   (******** Comparison of this number named p0 with p+1. **********)
   (*********** one knows that p0 < p+1 or p0 = p+1 : *************)
   (*************************************************************)
Elim (DIFFELNB_ConsArg_ConsArg13_le y y0 y1 y2 (S p) H4 p0 H5);Intros.  1680
```

```
   (*************************************************************)
   (*  DIFFELNB_ConsArg_ConsArg13_le:(t1,t2,t3,t4:quasiterm)(n:nat)      *)
   (*(DIFFELNB (list_var (ConsArg (ConsArg t1 t2) (ConsArg t3 t4))) n) *)
   (*       ->(n0:nat)(DIFFELNB (list_var (ConsArg t1 t3)) n0           *)
   (*                 ->({(le (S n0) n)}+{<nat>n0=n})                   *)
   (*************************************************************)

   (*************************************************************)
   (*************************************************************)
   (************************ Case p0 < p+1. *********************)
   (*************************************************************)
   (*************************************************************)

   (*************************************************************)
   (************ one applies the recurrence hypothesis H : **************)
   (* (q:nat)(le q p)->(u1,u2:quasiterm)                          *)
   (* (DIFFELNB (list_var (ConsArg u1 u2)) q)->(Unification u1 u2)  *)
   (*************************************************************)
Elim (H p0 (le_S_n p0 p a) y y1 H5);Intros.
2 : Auto.
(*Apply UnifConsArgfail1*)

   (*************************************************************)
   (******** the unifier f of f and y1 is either the identity, ************)
   (***************** or different of the identity : ******************)
   (*************************************************************)
Elim (ident_or_notS (ConsArg y y1) f o);Intros.

   (*************************************************************)
   (************************ Case f non identity ********************)
   (*************************************************************)
Elim a0;Intros x H6.

   (*************************************************************)
   (********* one supposes (Unification (Subst f y0) (Subst f y2)) ******)
   (*************************************************************)
Cut (Unification (Subst f y0) (Subst f y2)).
Intros H7;Elim H7;Intros.
Apply two_succes with f f0;Auto.
Apply one_only_succes with f;Auto.

   (*************************************************************)
   (********* proof of (Unification (Subst f y0) (Subst f y2)) : *********)
   (********* 1) (list_var (ConsArg (Subst f y0) (Subst f y2))) *********)
   (********************* counts x1 different variables. ****************)
   (********** 2) one applies the recurrence hypothesis H **************)
   (********** 3) for this. one uses the fact that the unifier f *********)
   (********** makes the number of different variables decreasing. ******)
   (*************************************************************)
Elim (DIFFELNBor (list_var (ConsArg (Subst f y0) (Subst f y2)))).
Intros p1 H7.
Apply (H p1);Auto.
Apply le_S_n.
Apply (f_n_id_minus y y0 y1 y2 (S p)) with f;Auto.
   (*************************************************************)
   (***** f_n_id_minus:(t1,t2,t3,t4:quasiterm)(n:nat) ********************)
   (* (DIFFELNB ***************************************************)
   (*** (list_var (ConsArg (ConsArg t1 t2) (ConsArg t3 t4))) n) ********)
   (* ->(f:quasisubst)(idempotent f)->(over f (ConsArg t1 t3)) ********)
   (* ->(under f (ConsArg t1 t3)) ***************************************)
```

```
   (* ->(<var>Ex([x:var](~<quasiterm>(V x)=(f x)))) ***************)
   (* ->(n0:nat)(DIFFELNB  ************************************)
   (********** (list_var (ConsArg (Subst f t2) (Subst f t4))) n0) *******)
   (* ->(le (S n0) n) ****************************************)
   (*****************************************************************)
Exists x;Auto.

   (*****************************************************************)
   (**************** Case f = identity extentionnelly ****************)
   (** 1) eq_V_stab3 reduces the problem to (Unification y0 y2), **********)
   (** 2) (list_var (ConsArg y0 y2)) counts p0 different variables, ********)
   (***** 3) si p0 < p+1, one applies the recurrence hypothesis H *******)
   (***** 4) si p0 = p+1, one applies the recurrence hypothesis H1. *****)
   (*****************************************************************)
Apply eq_V_stab3 with f;Auto.
Elim (DIFFELNBor (list_var (ConsArg y0 y2)));Intros p1 H6.
Elim (DIFFELNB_ConsArg_ConsArg24_le y y0 y1 y2 (S p) H4 p1 H6);
Intros.
   (*****************************************************************)
   (* DIFFELNB_ConsArg_ConsArg24_le:(t1,t2,t3,t4:quasiterm)(n:nat) ****)
   (* (DIFFELNB ***********************************************)
   (* ***(list_var (ConsArg (ConsArg t1 t2) (ConsArg t3 t4))) n) ******)
   (* ->(n0:nat)(DIFFELNB (list_var (ConsArg t2 t4)) n0) ***********)
   (* ->({(le (S n0) n)}+{<nat>n0=n}) ***************************)
   (*****************************************************************)
Apply (H p1 (le_S_n p1 p a0) y0 y2);Auto.
Apply (H1 y2 );Elim b0;Auto.

   (*****************************************************************)
   (*****************************************************************)
   (********************** Case p0 = p+1 ****************************)
   (*****************************************************************)
   (*****************************************************************)

   (*****************************************************************)
   (******** 1) one applies the recurrence hypothesis H0 ***********)
   (********** which proves (Unification y y1), *********************)
   (********2) one proceeds after exactly like above. ***************)
   (*****************************************************************)
Elim (H0 y1);Elim b;Auto;Intros.(*Apply UnifConsArgfail1*)
Elim (ident_or_notS (ConsArg y y1) f o);Intros.
Elim a;Intros x H6.

   (*****************************************************************)
   (********* one supposes (Unification (Subst f y0) (Subst f y2))******)
   (*****************************************************************)
Cut (Unification (Subst f y0) (Subst f y2)).
Intros H7;Elim H7;Intros.
Apply two_succes with f f0;Auto.
Apply one_only_succes with f;Auto.

   (*****************************************************************)
Elim (DIFFELNBor (list_var (ConsArg (Subst f y0) (Subst f y2))));
Intros p1 H7.
Apply (H1 p1);Auto.
Apply le_S_n;Apply (f_n_id_minus y y0 y1 y2 (S p)) with f;Auto.
Exists x;Auto.

   (*****************************************************************)
Apply eq_V_stab3 with f;Auto.
```

```
Elim (DIFFELNBor (list_var (ConsArg y0 y2)));Intros p1 H6.
Elim (DIFFELNB_ConsArg_ConsArg24_le y y0 y1 y2 (S p) H4 p1 H6);
Intros.
Apply (H p1);Auto.(*Apply Apply le_S_n*)
Apply (H1 y2 );Elim b1;Auto.
Save proof_unif.

Goal (t,u:quasiterm)(Unification t u).
Intros;Elim (DIFFELNBor (list_var (ConsArg t u)));Intros n0 p;
Apply proof_unif with n0;Auto.                                    1810
Save unif_proof.
```

ISSN 0249-6399